UNIVERSITY OF OSLO
Department of Informatics

Virtual User Simulation

Master thesis

Ellef Thurmann

23rd May 2005

Acknowledgements

The author wishes to thank his supervisor Hårek Haugerud for help and guidance, and all of you who have made comments, shown interest and given inspiration.

Abstract

This project is about human–computer interaction, consisting of two parts. The first part is to analyze human behavior through the observation of machine/system behavior – solely through network traffic. By observing this in a multi user system – what they do and when they do it, this thesis aims to see if distinct features of the population can be found and later reenacted. This will be achieved through observing network traffic, gathering it in a structured way and characterizing it. Humans and their computer workstations are the principal factors in making the traffic that is going to be measured.

The second part is to simulate the behavior and network traffic observed with virtual users – programs which behave similarly to their real counterparts. This is done with scripting, and results in a configurable tool which can simulate human behavior as seen from the network. Humans are thus eliminated in favor of computer simulated virtual users. The virtual users will try to generate network traffic with the same type, amount, and characteristics as observed in the first part, in a simulation with a hitherto unprecedented level of realism. Ideally, if this is repeated, it will form an idempotent loop ending up still looking like the original system with real users.

Contents

Chapter 1

Preface

This work represents Ellef Thurmann's Master Thesis in Network and System Administration at Oslo University College and Oslo University, written between January 1st and May 23, 2005.

Chapter 2

Introduction

When a computer system is about to receive new users, or a new service is about to be launched, uncertainty is introduced to the system. The uncertainty lies in how the system will behave in the modified state or environment – which may be that of added users, new services or new configurations. It is of great value for an organization to be able to predict what will happen and how it will affect the system. It is also important to keep the system in accordance with the policy stated service level agreement[1] while the organization's system changes, as is normal when it grows or new services are put to use.

This thesis will primarily try to provide a way of telling or estimating how the system will behave before actual deployment with real users occurs. It will investigate a method for experiencing the computer system when the number of users is increased to an arbitrary number. This means the same as to check the predictability of the scalability of the system with regards to user volume growth.

The way this is investigated, is by creating virtual, computer–controlled users that behave in the same manner as real life users, as observed from network traffic made when people use services like browsing the web and sending email.

2.1 Central questions

The central questions this thesis seeks to give an answer to are:

- What does a normal day of user generated network traffic look like?
- In what way can it be found out if a system solution that is operating in a production environment scales[2] when additional users enter the system?
- In what way can a tool be devised to make it possible to predict the scalability of a system?
- In what way can user generated traffic be regenerated to such a degree that on average, there is no clear distinction?
- Can this be a tool for estimating when modifications of existing configurations are needed?

[1] SLA

[2] i.e. is able to cope with the added load

2.2 Some possible benefits

It may seem like an oxymoron – creating users to counter the need for users to test the system. Nevertheless, it carries some distinct advantages over doing it "by hand" – letting ordinary users be the test pilots of the new services.

- It can be seen, how the new system configuration, server or existing system fares when the equivalent of any number of current users generated traffic is introduced in addition to what's already there. In other words, provoking a harbinger of future bottlenecks[3] to come, e.g. seeing how the system deals with added users before they are introduced in reality. This is essentially checking the scalability of they system.

- Make honey pots seem more like real computers that are in use. Honey pots are normal computers[4] in disguise, monitored for possible virus/hacker attacks/intrusions. Usually, they are not used for production work, but as passive alarms waiting to be triggered if something suspicious happens to them. By having virtual users causing realistic use of such a server, the disguise is further improved.

- Test the system for how it will behave and discover bottlenecks before actual harm may occur in a given scenario. This is equivalent of letting an Olympic tournament system run with simulated contestants and simulated service users to check for capacity, errors, bugs or flawed configurations.

- By deploying virtual users in large numbers, it could help making it possible to predict and prevent a failure such as breaking contracted service levels when e.g. the latency of the network links is too high. This is called proactive system administration - doing necessary work in advance of an error. The opposite is to do reactive system administration, which is to fix the problem after it has occurred.

- Provide a tool for the education of system administrators. Since there are intrinsic obstacles involved in getting ordinary humans to pretend they are users in a computer system made for educational purposes by students, it is far better to have simulated users that can be configured and toyed with on demand. This way, new configuration and services can be tested and evaluated, possibly using virtual machines which would further abstract the system administration learning process.

- Make testing more efficient by using machines to do what otherwise would have been done by humans. So in the end, humans, or system administrators, get to spend more time doing more fun things like playing games and socialize[5].

- Impede testing servers into a kneeling state of no response to show potentially unforeseen consequences. This would be disaster simulation for the benefit of disaster preparedness. The disaster may arise from such things as a deadline - where productivity soars to unprecedented heights, leaving an extraordinary toll on system services like none before.

[3] a place which cause congestion
[4] although often configured as a server, since they're more of an alluring bait.
[5] Or hurrying off to fix just one more thing..

2.3 Limitations

This is not about which system, configuration, scheme, architecture or network topology is best given arbitrary conditions and purposes. That is beyond the scope of this thesis; the aim is to develop a tool that can help making this possible. It follows the UNIX way of making small tools for specific tasks that can be joined together with other tools to perform bigger, more complex tasks.

Chapter 3

Background

3.1 About Oslo University College

Oslo University College, or OUC for short, is the largest university college in Norway with more than 10 000 students. It provides studies in many areas such as journalism, nursing, chemistry and computer science to name a few. The institute of engineering, or IU[1] for short, is separated from the rest, including the data network. IU is organized under its own subdomain, iu.hio.no and has its own administration. This is where the bachelor (and some master) degrees in electrical, chemistry, construction and computer science is. It has about 1675 students (as of 28 of May, 2005).

The students at IU are provided with file storage, email service and web services in addition to subject relevant services and programs. Each student have their own login and password to mostly MS Windows workstations[2]. Each have their own email account accessible through IMAP, and home directory on a network file server (running Novell Netware) for MS Windows workstations. All workstations have access to the Internet.

This was the setting of the VSIM[3] experiment. Before more light is shed on the VSIM experiment, some illumination on relevant technologies is needed.

3.2 Protocols and services

To be able to grasp the project for someone not intimately acquainted to computer terminology, it is necessary to provide some background and terminology that is used throughout the rest of the thesis. The topics are only briefly going to be discussed, so for more in–depth knowledge, the reader is advised to read an instructive computer network book such as [1].

3.2.1 Network

A local area network (LAN) is a network of computers and hardware connected in a small geographical area, like an office or building. By using a LAN, users are able to use services such as mail and web browsing, as well as sharing equipment such as printers more efficiently than if no network were available. Such LAN networks are often comprised of one or more servers connected to several workstations, which are the computers ordinary users do their work on.

[1] Ingeniør Utdanning – Institute of Engineering
[2] some Debian GNU/Linux workstations are also available
[3] Virtual User Simulation

A common medium which connects these components are 100 Megabit Twisted Pair Category 5 (100Mb TP CAT-5) cabling. 100 Megabit means the cable can sustain a data rate of 100 000 000 bits per second, roughly equal to 11.92 Megabytes per second theoretically[4]. Such cables are now generally connected point to point between computers and switches. The switch is a device that can accept several inputs and direct them to one or more outputs; so that several cables which in turn are connected to computers, can be connected to a switch, just like a bus terminal connects several bus routes.

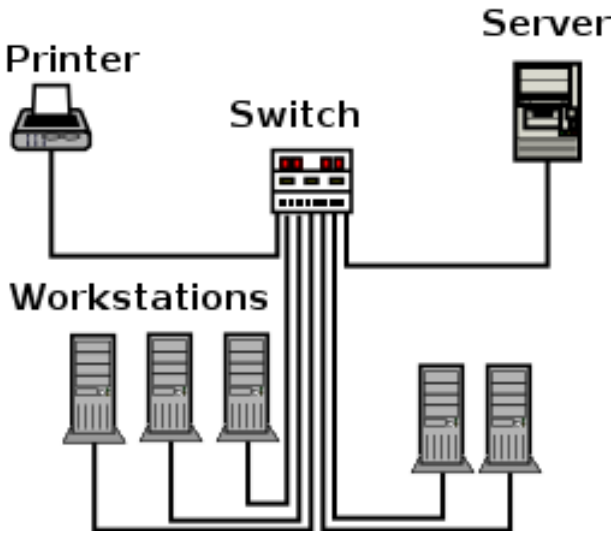

Figure 3.1: A little LAN with workstations and a server connected through a switch.

A switch can also support Virtual Local Area Networks (VLANs). A VLAN is the same as a LAN except that the administrator can choose which computers should be in which VLAN. This is configured on the VLAN capable switch. One can assign arbitrary ports on the switch to any VLAN that has been configured. The VLANs then become essentially the same as a LAN.

The switch operates at layer two of the TCP/IP stack, which is a layered approach for the transmission of data. The first layer is the Physical Layer. There, the electrical signals are defined, such as voltages and medium types. Layer two decides how the data is encoded, such as with Manchester encoding if following the Ethernet standard, and routed within the LAN. A switch is therefore in this context an Ethernet switch. Layer three is the IP layer, which makes the data routable over the internet. Layer four is the transport layer, taking care of the transmission, e.g. making sure each packet arrives in order. Layer five is the application layer which can be any service protocol such as HTTP, SMTP etc. For more on this subject, read e.g. [1].

3.2.2 TCP sessions – a brief introduction

A TCP session is a standardized way of communication over an IP network. It is connection oriented (as opposed to UDP, where no preparations are done before a data packet is sent), meaning that some packets are exchanged before the first data packet is sent. This is called a three–way handshake, and essentially opens up ports on either host, preparing them for reception/transmission. A port is like a

[4]But to transfer anything over the line, overhead and error checking reduces this somewhat depending on the protocol used.

Level	Layer Name	Example
1	Physical	TP cable
2	Data	Ethernet frame
3	Network	IP, IPX
4	Transport	TCP, UDP
5	Application	HTTP, NCP, IMAP

Table 3.1: The TCP/IP protocol stack

numbered gate where data can pass in or out. Each service usually runs on one or more designated TCP/UDP ports.

Each TCP packet contains a sequence number so if one goes awry, or arrives out of order, the error can be corrected. This is in contrast to UDP where there is no telling if a packet is lost/out of order. A TCP connection also has to be torn down in much of the same manner as it is set up.

3.2.3 HTTP

Underpinning what's commonly known as "web surfing" or "browsing the internet", is HTTP, the Hyper Text Transfer Protocol. This is an application protocol, usually run over TCP on port 80. Application protocols are the same as services in the context of this thesis.

3.2.4 NCP and SMB

The standard protocol for file and printer services mostly used for MS[5] Windows workstations, is CIFS[6]. MS workstations typically connect to a share[7] offered by a MS Windows server. These are also called SMB[8] shares. A freely available implementation made for Unix like operating systems called SAMBA is available from http://www.samba.org. An commercial product and a competitor to Samba/CIFS is Novell's NetWare services which use their own protocol, NCP[9], for file and printer services.

NCP manages access to the primary NetWare server resources. It makes procedure calls to the NetWare File Sharing Protocol (NFSP) that services requests for NetWare file and print resources. NCP is the principal protocol for transmitting information between a NetWare server and its clients.

NCP handles login requests and many other types of requests to the file system and the printing system. It is a client/server LAN protocol. At the server, NCP requests are received, unpacked, and interpreted.

NCP services include file access, file locking, security, tracking of resource allocation, event notification, synchronization with other servers, connection and communication, print services and queue management, and network management.

NCP uses the underlying Internetwork Packet Exchange Layer Services (IPX[10]) or TCP/IP which NetWare versions after NetWare 5.0 can use.[11]

After some observation of the network traffic at IU/OUC, it was seen that TCP was mostly used as a transport protocol for NCP.

[5] Microsoft
[6] Common Internet File System
[7] An object, usually a filesystem directory that is accessible over network
[8] Server Message Block
[9] The Novell NetWare Core Protocol
[10] Internetwork Packet Exchange Layer Services, an OSI layer three protocol
[11] From http://www.networkdictionary.com/protocols/ncp.php

3.2.5 SMTP

SMTP is the Simple Mail Transfer Protocol which controls the transmission of email between MTAs. MTAs are Mail Transfer Agents, commonly known as email servers (like Microsoft Exchange or Postfix). A MUA (Mail User Agent), or client, connects to a MTA, delivers an email, the MTA contacts and delivers it to the destination MTA, where the recipient can read it with a MUA through e.g. IMAP. SMTP is an application protocol running on TCP port 25.

3.2.6 IMAP

IMAP is an access method for polling an email account, to retrieve and organize the emails there. The emails will remain on the server as opposed to POP, where each email is removed from the server when the client has retrieved it. IMAP is an application protocol running on TCP port 143.

3.3 Service testing

These services are all common and important ingredients in computer systems. It is valuable to learn how much of these are used on a system, and how much of it the system can handle before it slows down, as pointed out in section 2.1, page 5. An underlying theme is the scalability of system performance, and how to go about measuring it.

Benchmarking is often used in situations where one would like to measure the maximum performance of a particular system or configuration. What has become particularly popular in the last years is to benchmark 3d graphics accelerator cards from various manufacturers to determine which one gives the highest frames per second in the latest and greatest 3d game. This is the same as the approach taken to measure computer system server performance. A typical benchmarking suite will test a few parameters where the most common one is the number of served requests per second. This can for instance be HTTP service requests to a web server, or emails sent for a mail server (e.g. 10 emails per second). This is a well known way to find out the maximum rate of requests the current configuration can handle predictably.

This model is perhaps a bit simple when one considers that these benchmarks don't to a high degree reflect the current and would–be usage of the system. Normal computer users don't act like a benchmarking or stress testing application over time, but benchmarking does give an important indication of performance.

A tester (usually a system administrator) can for some configuration check to see how many requests per second is serviced on average, the maximum and minimum number of requests served, as well as calculating the standard deviation to see how much it varies. The tester then compares this result with the maximum requests per second as reported by the benchmarking or stress testing application, to see how large a margin there is for scalability. A large gap is likely to calm him down since it allows for additional service usage.

An argument against this is that the tester won't know how many more users that gap in potential performance constitutes. Also, the benchmarking will have to be performed when the system is not needed by other users, since it will use up the capacity to calculate maximum performance. Benchmarking is still a workable approach for testing a single service, but few describe the performance of the total system when it consists of more services. If e.g. the SMTP server is benchmarked to serve up to 1500 requests per second, it may not be that fast when at the same time, the IMAP server has a run for its money.

To get another clue towards, and get nearer to describing the correct performance of a system comprised of more than one service, this thesis tries to look towards other methods. By first examining the network traffic, an overview of what the traffic and usage patterns look like for a networked computer system with human users is obtained. This serves as a base for creating a description of the system usage. In the next part of this approach, this description will serve as a base for generating human network service traffic and simulating user system usage.

This approach is the starting point for a look through related work researched by others.

3.4 Literature survey

3.4.1 Introduction

This survey is about the four ingredients of the thesis – namely system behavior, system observation, system testing and regenerating network traffic. Each topic is described separately, and relevant papers presenting an overview of their status in each respective field is provided.

3.4.2 System behavior

First off when analyzing a system is making a model. System administration has often been perceived as a practical, hands–on profession, but there is a clear need for theoretical approaches to complement that [2]. By using theoretical models, it is easier to form a solid foundation on which to build a policy. The policy is like the rule–set of the system, and it is therefore necessary to do ones best to uphold the integrity and validity of the policy – hence it is important to know if the system policy could be compromised due to unpredictable scalability again due to user behavior and user quantity increase.

The next question would then be how to model and predict such activity. A practical approach on how to simulate real user behavior is given in [3]. A computer system is modeled by individual users who have separate needs for resources like processes and login times. During this simulation, the simulated users make decisions with probabilities which depend on the time of day and on the character of the user. This enables the reproduction of large scale behavior measured at real computer systems as well as predicting the behavior of systems when varying the number and characters of the users.

What it also shows is that it is possible to reproduce usage patterns in large multi–user systems by using relatively simple parameters. Users are characterized into different types, whether he is always in, system user, or standard user, when he normally logs in and out, and if and when he is likely to start processes. This is modeled by probabilities, and is carried out by Monte Carlo simulation.

3.4.3 System observation

In previous work, it has been shown how the average behavior of systems of computers and users can be approximated by a blend of statistical models and thermodynamical ideas [4]. That work allows one to form a mathematical model of computer systems which can be used as a basis for modeling system administration. Many macroscopic system variables are periodic in nature [3], and stochastically distributed around stable averages, modulated by periodic variation such as those occurring during the transition between day and night, working hours and free time.

One method of observing this kind of behavior and it's variations is to categorize the network traffic generated. In [5], an algorithm for inferring network traffic into

categories is devised. The authors have recognized the need to better know what traffic is on the network, and by using their algorithm, they are able to further analyze the contents of IP traffic. The method used is called multidimensional traffic clustering.

There are a dozens of applications for categorizing network data, mostly for layer three and four in the TCP/IP stack[12], such as Ntop, MRTG and Munin. These work on a statistical level, and tells first and foremost about the quantity of network traffic, what kind of traffic it is seeing (ICMP,IP,UDP,TCP), and in Ntop's case also include functionality to see the current distribution of a selected few layer five protocols, who is causing it and when.

Also working on the lower layers are TCPdump, Argus, tethereal and ethereal. Ethereal is a graphical packet analyzer which has the ability to summarize data into a protocol hierarchy with traffic statistics. Tethereal is the console based equivalent. It can also measure round trip times for selected protocols like CIFS/SMB. Since tethereal is console based, it can also be used in scripting to further leverage its quite extensive protocol matching capabilities, to further analyze things like packet size, delay and so on. A selection of these applications are put under scrutiny in section 3.5, page 17.

Measuring service delay has also been researched on a grander scale, on the wide area network[6]. The motivating question for this project was why the Web was so slow. They conclude they need to implement the experiment on a larger scale, with more analytical content to achieve further answers. But in the process, they discover an interesting fact about their server. They show that (for their server) the main effect of server load on typical transfers is to delay the first data packet sent. In addition they show that in many of the experiments, servers under high load suffered significantly less packet loss than those under low load: When the network was heavily loaded (i.e., packet loss rates were high), it was not uncommon for a heavily loaded server to show better mean response time than a lightly loaded server. Their measurements suggested that this may have been because heavily loaded servers often show lower packet loss rates, and since packet losses have dramatic effects on transfer latency, this can reduce mean response time.

Another area in which traffic categorization and analysis is used, is in anomaly detection. In [7], one of the primary tasks of network administrators is identified to be monitoring routers and switches for anomalous traffic behavior such as outages, configuration changes, flash crowds and abuse. In this paper, the focus is on precise characterization of anomalous network traffic behavior. They seek to use statistical methods to distinguish and portray similarities among anomalies. While a variety of commercial and open source tools have been developed to assist in this process, these require policies and/or or thresholds to be defined by the user in order to trigger alerts. The better the description of the anomalous behavior, the more effective these tools become.

Their stated goal was to identify precisely the statistical properties of anomalies and their invariant properties if they existed. At the time of writing they were building an archive of anomalies based on IP traffic flow measurements taken from their border router placed on their campus network, and an early stage of applying various statistical analysis techniques to the data was reached.

Remco Poortinga et. al. wrote a paper discussing its way to find which students were using their quota of network traffic to the fullest in the University of Twente campus net[8]. From measurements they learned that their method of reading IF–MIB[13] counters within the access switches was not really adequate to predict the load that an individual student placed on the shared backbone link connecting

[12] see section 3.1, page 10

[13] An interface which measures traffic volume for each host, for each port the host has connected to.

the CAMPUSnet to the external world. To determine how much load individual students put on the backbone link, that link would be measured directly via tools like NeTraMet. Their measurements also showed that the vital few and trivial many (also known as the Pareto Principle or the 80–20 rule) applied to the shared backbone link and the amount of traffic sent on the CAMPUSnet(see figure 3.2).

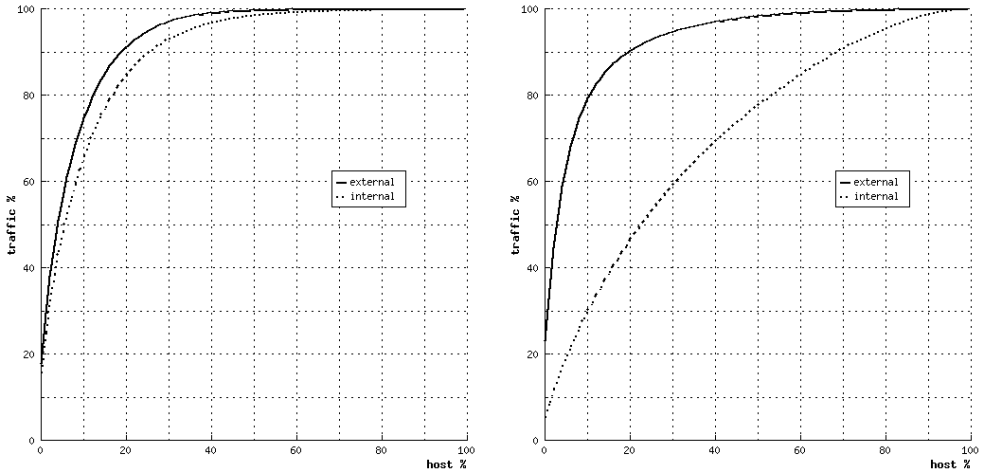

Figure 3.2: Percentage of hosts responsible for percentage of traffic sent(to the left), and received(to the right).

If the campus link were perfectly shared, a graph plotting number of hosts and accumulated amount of external traffic would show a straight diagonal line at a 45 angle for the external traffic. However, the lines show that for external traffic, between 10% and 15% of the hosts are responsible for 80% of the traffic in either direction. For internal traffic the distribution is very different between traffic sent and received. The graphs show that a lot of traffic is sent by a relatively small percentage of hosts but that reception of traffic is distributed more evenly over a larger percentage of the hosts, suggesting that a limited number of hosts serve a lot of others locally.

However, this inequality rule does not apply to the amount of traffic received locally on the CAMPUSnet. Furthermore, they found that NeTraMet, running on an ordinary PC, was capable to reliably capture and analyze all packets flowing over the campus link. Depending on the number of rule sets in use; it could easily handle 60 thousand packets per second, which is equal to roughly 350 Mbit/s.

As seen, there are various methods of observing the network traffic where many choose to develop their own software for network traffic observation.

3.4.4 System testing

This is the first step towards testing the service level, scalability and predictability of distributed software (networked applications) in a typical LAN setting. It is not about measuring latency of typical binary programs like spreadsheets and office tools.

Others have investigated the scalability of various parts of a system such as the Andrew distributed filesystem (AFS)[9]. This paper mainly tells about the experiences running AFS, and how it seems to scale and work well. A comparison with NFS shows that AFS scales better and has got more convenient benefits such as quotas, convenient mounting and easy backup aside from running in userland

mode. When running in userland mode as opposed to being a part of the running kernel, it can be regarded as more safe.

In [10], network server performance demands are measured. They do this by generating realistic HTTP client requests. Unfortunately, accurate generation of such traffic in a test bed of limited scope is not trivial. In particular, the commonly used approach is unable to generate client request–rates that exceed the capacity of the server being tested even for short periods of time. The rate of generated requests never exceeds the capacity of the server. To generate a significant rate of requests beyond the capacity of the server, one would have to employ a huge number of client processes.

This is also a challenge when making virtual users which should produce bursty traffic. One way of circumventing this possible limitation is to simply run the simulation from more than one host at the same time.

The paper proposes and evaluates a new method for Web traffic generation that could generate *bursty* traffic, with peak loads that exceed the capacity of the server. The model, Scalable–Client, had two key ideas. To (1) shorten TCP's connection establishment timeout, and (2) to maintain a constant number of unconnected sockets (simulated clients) that's trying to establish new connections. S-Clients enabled the generation of request distributions of complex nature and with high peak rates, something which wasn't possible using a simple scheme for request generation.

Other problems found were that in particular, the simple method did not model high and variable WAN delays which are known to cause long SYN–RCVD queues in the server's listening socket. Also, packet losses due to congestion were absent in LAN–based test beds.

Also, client side CPU and memory contention was likely to arise. Eventually, a point is reached where the bottleneck in a Web transaction is no longer the server but the client. The primary factor in preventing client bottlenecks from affecting server performance results is to limit the number of simulated clients per client machine.

The paper also makes an useful explanation of how web browsing works down to minute detail.

Another paper closely related is describing the Dynamic Workload Generator[11]. This describes an application which generates synthetic network load for use in load–balancing experiments. The challenge was to make the generated workload not only mimic the highly dynamic resource–utilization patterns found on today's distributed systems but also behave as real workload does when test jobs are run concurrently with it. It was capable of replaying network wide patterns recorded earlier, but the main focus was on describing the Synthetic workload generator. It was implemented inside the kernel and controlled CPU, memory, disk, and network resources. Some problems with only using replays of old user traffic was identified as well.

An introduction to the fundamentals of the concepts and features of network benchmarking is given in [12]. A benchmark is an evaluation technique that acquires performance measurements on a system to be used as reference to assess the systems. The results from benchmark tests help in making hardware and software design decisions and also help in purchasing network system components. Overview and test usages of several benchmarking applications are given.

As for testing multiple services, there are many ways they can be tested. As Dan Clein put it in his talk "Flying Linux" at LISA04[14], loading and using the different modules of a system, e.g. that of the kernel, constitutes so many combinations to test that it would take ages if even one combination every second could be tested.

[14]The Large Installation System Administration conference, 2004

For 32 modules, it would be equal to:

$$32!\,\text{modules} \times 1\,\text{second/module} = 2.63 \times 10^{35}\,\text{seconds}$$

which is

$$32!/(60 \times 60 \times 24 \times 365) = 8.34 \times 10^{27}\text{years}.$$

So it takes a bit of time to test a system based on more than a fistful of modules for every combination – it's practically unviable for systems with growing modularity as it grows with *faculty* growth rate, which is even worse than exponential growth. What this means is that one must try to test for the combinations that may, or *is likely to occur*, in a system. Thus, one of the aims of this project is neither to test only one service, nor testing them all, but several important ones[15], at the same time, potentially uncovering unknown side effects.

3.4.5 Regenerating network traffic

A possible approach to regenerate traffic is by using the a–b–t model[13]. It was an empirically–based approach to synthetic traffic generation. Starting from a trace of TCP/IP headers on a production network, an application–independent, source–level model was constructed for all the TCP connections observed in the network. The model, a set of a–b–t connection vectors, was provided to the Tmix traffic generator that replayed the connections and reproduced the application–level behaviors observed in the original network.

A source–level model of the data exchange dynamics inside a TCP connection was built. It was able to model the complete mix of TCP connections seen on a link. This was an advance compared to previous models which only modeled the traffic from a single application (e.g. HTTP). The model encompassed two cases. The first consisted of *(a, b, t)* connections where a was the message sent one way, b was the response and t was the delay before the response was sent.

Concurrent connections where endpoints exchange data continuously, was also modelled by using variables a and t_n, where a was one endpoint and t_n was the times n it sent. The aim was to produce synthetic traffic as real as possible, and the early results looked promising according to the paper.

In addition, issues involved in generating synthetic traffic based on measurements of network links was also done. A case study was presented of reproducing important features found on two Internet links an OC-48 link in the Abilene backbone and a 1 Gbps Ethernet link connecting a large public university with its ISP, in a laboratory testbed. They claimed that an aspect of their approach was a trace–driven method for source–level traffic generation that (1) modeled the mix of application traffic found on a network link and (2) performed a source–level "replay" of the traffic in a simulation or testbed network. Through this work they hoped to demonstrate ways in which the level of realism in networking simulations can be improved.

A somewhat different approach is to make synthetic HTTP traffic[14]. A new model of HTTP 1.0 and 1.1 source traffic as it appears in aggregate on an access link is presented. The model is used to generate synthetic source traffic for a network modeled either by a network simulator such as ns or a hardware testbed.

The model derives from a large–scale empirical study of web traffic on the two access links that provide Internet connections for Bell Labs and for the University of North Carolina at Chapel Hill. The model was novel in that it expressed web

[15] See section 4.1, page 22

traffic as a sequence of TCP connections in which aspects of each connection were described by values of stochastic source–traffic variables. The model consisted of a collection of statistical models that determined the stochastic properties of the variables.

Another way of generating synthetic network traffic has also been researched[15]. The target was to generate realistic workloads for network intrusion detection systems (NIDS). It therefore didn't aim to put the stress on any services, but to test the NIDS by generating dud traffic.

The necessity of developing a model that captures the performance of products on the customer site was essential for any company to meet the rapidly changing needs of the customer[16]. The movement away from homogeneous server centric sites to decentralized heterogeneous environments was impacting the behavior of systems and the cost of servicing of the system in ways which were not yet fully understood. A process named DPP (Digital Product Performance) provided a mechanism to study the actual behavior of systems and to identify the factors which impacted this behavior. The DPP process consisted of four items; mainly, (1) Data collection on the customer site, (2) Data transportation from the customer site to the DPP group in Ayr, Scotland, (3) Management and storing of the data, and (4) Use of analysis software. The data was stored in a relational database, where it was analyzed for system availability and product reliability (e.g. operating system reliability).

The paper claimed the model to be flexible and to provide root cause analysis of the reliability and availability of a wide range of products and operating systems.

Checking the reliability and availability of services can also be done by simulation. Simulation is a method of creating a model of a proposed system or a way to deal with a real system with an imitation in order to study the behavior of the system under specific conditions. Using simulated models is cheaper and safer or even more possible than to make experiments with the real systems. In numerous areas, simulations have been producing various benefits [17].

3.4.6 Summary

The ideas on which this thesis is mostly built on, are probabilistic and Monte Carlo simulations as given in [3]. Since [3] gave good results, a corresponding method should have a good chance of equally good results.

3.5 Tools of the trade

To both observe and regenerate network traffic, a search into available programs that could be used was done. Since this thesis was done under a relatively short span of time (from January to May), time was saved by utilizing already existing applications in favor of developing new ones. There were many tools of various quality and functionality, so only the freely available, open source and those with a reputation of workability were chosen. In particular, [18] proved a good starting point and introduction to the various tools.

The tools which were found and used are described underneath. The numerous other tools that were found, tested but not used are for a large part not described since they are mostly irrelevant to the experiment itself. For completeness, and since many of them can be used instead of the ones chosen, interchangeable tools are included in addition to tools which can be helpful if additional types of observations are needed.

3.5.1 Programs

Most of the descriptions here have been borrowed from the respective products home pages.

- TCPdump – A program which dumps traffic on a network. It is used for network monitoring, protocol debugging and data acquisition. It is known as *the* packet dumper and it is perhaps the most widely used network traffic dumper available. TCPdump dumps packets in libpcap (library for packet capture) format, which is a very common format for storing packets. The network traffic is saved in files commonly known as *dump* files.

- TCPreplay – Replays packets captured in libpcap format. TCPreplay is often used to replay tcpdumped data while dumping it on the other end of the pipe. It's primarily aimed for IDS testing – so the early versions will first and foremost just generate the traffic, not making sure the traffic with responses are correct. By the time of this writing, it is still an early version.

- MRTG – The Multi Router Traffic Grapher (MRTG) is a tool to monitor the traffic load on network links. MRTG generates HTML pages containing graphical images which provide a LIVE visual representation of this traffic. But it only shows the amount of traffic, and doesn't split the traffic into different application protocols.

- Ntop – ntop is a network traffic probe that shows the network usage, similar to what the popular top Unix command does. Ntop is based on libpcap and it has been written in a portable way in order to virtually run on every Unix platform and on Win32 as well.

  Ntop users can use a web browser (e.g. Netscape) to navigate through ntop (that acts as a web server) traffic information and get a dump of the network status. In the latter case, ntop can be seen as a simple RMON–like agent with an embedded web interface. The use of

  – a web interface
  – limited configuration and administration via the web interface
  – reduced CPU and memory usage (they vary according to network size and traffic)

  make ntop easy to use and suitable for monitoring various kind of networks. Ntop is easy to like, but as with the Unix top command, it provides mostly current trends and not trends along a timeline, but pie diagrams instead. Also making it unsuitable for this experiment is that it doesn't have a console text mode output for further data processing by another program. Also noted is the fact that it identifies only a handful of the most common network protocols. See www.ntop.org for further details.

- Munin – Munin is a system to gather and graph all sorts of information. You can install a node on the various machines in your network, as well as on a central server. The nodes will know how to extract various kinds of information, such as load average and bandwidth usage, and will wait for the server to request these values. The output is in HTML format. The server can optionally send notifications if any of the values being observed moves outside of a specified range (and when they move back into it). It basically does much of the same as MRTG.

- Argus - the Argus suite monitors and gathers network statistics, and has capabilities to observe jitter, delay and throughput. However, it is not well documented and too few protocols were supported for the requirements of this project.

- Ethereal - a much used GUI[16] network analyzer. It was found to be unusable for dump files of some size (50 Mb and more), since it would eat up memory as fast as Takeru *the Tsunami* Kobayashi[17] eats hot dogs, feasting the swap file for desert and bringing the system to a grinding halt, refusing to respond to even the most innocuous of inputs.

- Tethereal - a console based version of Ethereal. Tethereal is capable of doing the same as Ethereal, and was thus the most advanced console based tool with regards to functionality known to the author at the time of writing. It can give a summary after each packet dump, displaying the aggregated traffic categorized into each service seen. This provides a way to generate graphs to see the traffic distribution.

  An undocumented shortcoming which was uncovered was that if a ring buffer with a specified number of files of more than 1024 was specified when dumping traffic, tethereal would only write to a maximum of 1024 files. However, if 0, for unlimited files, was specified, it wrote to more than 1024 files.

  Tethereal proved to be the one tool supporting the highest number of relevant protocols. A sample output shows how it identifies the protocol of a packet as Novell Core Protocol:

  ```
  1 17:05:55.193119 128.39.74.60 -> 128.39.75.8  NCP Service reply[Short Frame]
  ```

  It was assumed that tethereal worked by one or combination of 1) inspecting the packet payload, 2) its destination port, 3) its TCP session or 4) something else. Note that this is not critical to understand for the success of this experiment. It does however incur an uncertainty in how the packet count, or rather pseudo packet count, is derived.

  If 2) were chosen, it would be more inaccurate since a packet to that port would be classified as a service usage even though it was just a SYN or FIN packet, misconfigured or without data content.

- gnuplot – a program to plot graphs, equations and more. It can easily be scripted, and in contrast to XMGR, another graphing tool, it supports time on the x–axis.

- wget – a program which downloads web pages. It can be launched from a console, and can be configured to ignore robots.txt restrictions[18]. It also has options for recursively getting contents.

- SMBclient – a ftp like client to access MS Windows/SAMBA SMB/CIFS shares. There are free tools to mount Novell Netware / IPX Netware volumes, but not any proper servers. The decision was made to use the Samba suite (SMB/CIFS) as a replacement for NCP since no servers were found to be free and readily available for Linux.

  Samba is software that can be run on a platform other than Microsoft Windows, for example, UNIX, Linux, IBM System 390, OpenVMS, and other

[16] Graphical User Interface
[17] the current (2004) world champion hot dog eater with 53 1/2 in 12 minutes
[18] These are rules for web spiders / robots that traverse web pages

operating systems. Samba uses the TCP/IP protocol that is installed on the host server. When correctly configured, it allows that host to interact with a Microsoft Windows client or server as if it is a Windows file and print server.

On the other hand, Samba behaves a bit unlike other services. If one is logged in to a samba server with smbclient, the server, or the client, sends updates every other second or so. To prevent this from diluting the measurements of traffic that was going to be taken, smbclient was chosen instead of using smbmount which mounts a share just like a cd-rom, until it is unmounted. This is however, the normal way a workstation operates: When logging in, after the username and password is authenticated, the client receives the profile of the user, and the client usually mounts a share from the server, like HOME on H:.

- timeout – A program which starts another program and lets it run for a specified number of seconds before it is terminated with a specified signal. It was necessary to stop tethereal by using timeout since on some occasions, when tethereal was dumping to unlimited dump files, it couldn't be configured to stop by itself.

3.5.2 Perl modules

The implementation of the methods to be used for both capturing and regenerating network traffic were written in Perl, as wrappers to glue the different programs together. Several Perl modules chosen alleviated the development and lessened the need for external programs where used:

- Mail::IMAPClient – An IMAP Client API (Application Programming Interface, a set of functions and objects to use when programming with IMAP). This module was chosen to interact with IMAP servers.
- HTTP::Lite – Lightweight HTTP implementation which was not chosen since it didn't support HTTPS.
- WWW::Mechanize – Handy web browsing in a Perl object. Not used in favor of wget, but potentially usable since it may provide a way of capturing web pages without involving the client disk (which may be a performance delimiting factor).
- Expect - This expect module makes it possible to open a two way communication with a console program, so you dont have to physically be there to interact with the program. It is designed to eliminate the need for a human to feed input into programs prompting or requiring input when it is running. Expect makes it convenient to use SMBClient and other console based tools that require interactivity.
- Net::Pcap – This module was not chosen since it would require the parsing of headers, reconstructing fragments, checking for port number or parsing for application data, assumed to take more time and effort compared to using Tethereal.
- Mail::Sender – To send an email. It proved to be an straight–forward way to send an email:

```
use Mail::Sender;
       $sender = new Mail::Sender
        {smtp => 'mail.yourdomain.com', from => 'your@address.com'};
```

```
$sender->MailFile({to => 'some@address.com',
 subject => 'Here is the file',
 msg => "I'm sending you the list you wanted.",
 file => 'filename.txt'});
```

Chapter 4

Methodology

The approach taken to answer the central questions[1] was split up in two parts: The first part was to observe and analyze the network traffic, and the second was to regenerate the observations of part one.

Common to both is that they start off by drawing data from captured network traffic (a dump) in libpcap format. Both tasks were done by a script, so it would 1) be easier to develop, 2) be reproducible, and 3) have fewer possibilities for introducing errors.

4.1 Observing and analyzing traffic

It was necessary to find out which services were the most *important and largest* (in terms of traffic volume) on the network. The most important ones were the ones that were the most crucial for the users ability to successfully do their job, and the largest were those that caused the majority of the traffic. These should optimally be the same services.

After some discussion and observation of the organizations use of networked services[2], the most important ones *for common users* were found to be file, email and web services. They didn't always generate the largest volume of network traffic (multicast, P2P and FTP were also observed in abundance), but they were arguably the most important ones for common users to be able to do their job. For members of the computer staff, other protocols (such as SSH) were as important. But they were not the target user behavior which were sought to be replicated.

By observing the organizations workstations applications, the applications used for these services were identified. Microsoft Internet Explorer was available for web surfing. Web surfing is here defined as HTTP service use, surfing on the internet and the organizations own pages. File service is the use of a networked storage space to facilitate roaming use of workstations – one can log on anywhere, and still have access to private files and settings since they are accessible through the network. Novell Netware through the NCP protocol was used for this. In addition, SMB was also observed, which originated from MS workstations mounting SAMBA shares on a Linux server. Since both were file service use, they were treated as the same throughout the simulation. Sending and retrieving email could be done using MUAs such as Mozilla Thunderbird on the workstations, and these also use SMTP to send emails to the MTA which is the central email server, for further processing, and IMAP to retrieve incoming emails.

[1] see section 2.1, page 5
[2] see section 3.1, page 8

To gather the statistical characteristics of the traffic, several tools were considered, as previously explained. While many provided to be adequate to be read and interpreted by humans, they couldn't be easily and quickly transformed into something custom made computer scripts could parse and use for part one; observing and analyzing the data, and part two; generating the same traffic with the same characteristics.

As described in section 3.5.1 on page 18, after some experimentation, tethereal proved to be the best bet covering the greatest deal of the requirements for the project. Using tethereal, a method had to be devised to convert its output into informative graphs, showing the protocol distribution and relations with each other.

The method began by running tethereal to dump network traffic in intervals of a specified number of seconds. Each dump file was named after the time it was created, and the dumping ran for the duration of the interval. The timestamp part of the filename was used as x–axis tic labels on diagrams constructed of the data showing the distribution of the protocols over time. The x–axis was displaced by the specified interval of seconds, typically one or two minutes. However, as the difference would be small and the diagrams would mostly be plotted with hours or more as tic marks, making it negligible.

All this functionality was implemented in a Perl script called *capture.pl*.

4.2 Implementation of the analyzing part

4.2.1 Capture.pl

capture.pl dumps network traffic and plots the distribution of the different protocols (all layers) of the dump files in a time–series[3] plot. It is a Perl script requiring timeout, tethereal and gnuplot to work.

capture.pl generates this directory structures:

```
-rwx------    1     7.6K 2005-05-06 13:22 capture.pl
drwx------    2       48 2005-05-06 13:22 plcdir
drwx------    2     1.2K 2005-05-06 13:22 plotdatadir
drwx------    2     1.3K 2005-05-06 13:22 plotdir
drwx------    2      552 2005-05-06 13:22 txtdir
```

plcdir is where the dump files are dumped, txtdir and plotdatadir are where intermediate data is kept before plots are made of them in plotdir. capture.pl contains four functions, which are normally called in succession:

- getdata() dumps traffic and stores the traffic in an arbitrary number of files, depending on how long each capture should be. It has these arguments:
  - The interface to capture from.
  - The directory to save dump files.
  - The number of files, 0 for unlimited or up to 1024.
  - The capture duration in seconds for each dump file. Each dump file is therefore the packets seen in one interval.
  - The total duration in seconds of the dumping process. If the number of files is 0, tethereal must be stopped with timeout.
- parsedata() parses the dump files into text files with the aggregated traffic seen for each service. It therefore produces as many files as there are dump files. Parameters are:

[3] See [19] for more details on time–series.

  - The directory where the dump files are stored.
  - The directory to save the per dump text files.

- summarizedata() summarizes the per dump text files into per service text files suitable for reading by gnuplot. Parameters are:
  - The directory where the per dump text files are stored.
  - The directory to save the per service text files, ready for gnuplot.

- plotdata() plots the summarized per service text files. Parameters are:
  - The directory the per service text files are stored.
  - The directory to save the plots.

- plotdata2() plots the selected core services in a single plot. Parameters are:
  - The directory the per service text files are stored.
  - The directory to save the plot.

Thus, to do it all in one go, it could look like this:

```
getdata("eth1", "plcdir", 0, 60, 3600);
parsedata("plcdir","txtdir");
summarizedata("txtdir", "plotdatadir");
plotdata( "plotdatadir", "plotdir");
plotdata2( "plotdatadir", "plotdir");
```

The generated plots would then be found in *plotdir*. The script plots all protocols tethereal identifies. By editing the script it is possible to pass additional parameters to tethereal, like dumping traffic only for one IP, or one subnet, or filtering out responses from servers.

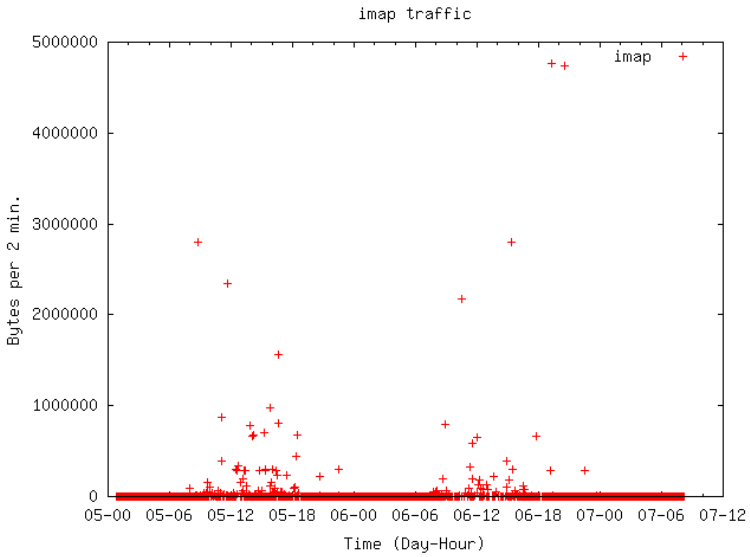

Figure 4.1: A sample plot, showing the IMAP distribution over two days. The scales must fit the observation, as this is an example of not doing too well.

4.3 Regenerating traffic

The method devised to regenerate the traffic was to implement a fluent transformation of a network traffic dump (libpcap format) by a generic algorithm that read and interpreted the dump, into data structures containing probabilities for using each service. These probabilities were used for checking when synthetic service usage should be performed by virtual users. The virtual users were implemented as processes.

Two different models for regenerating traffic were devised. The first model, model one, was only going to simulate one hour's worth of network service usage. It was based on observing the sequence of network service usages a user would do, as seen on the network.

The second model, model two, would simulate 24 hours worth of network service usage. It would be based on observing how the network service usage varied over time, as seen on the network.

4.4 Assumptions and compromises

There were several obstacles which appeared during the development of either model that had to be addressed.

4.4.1 What caused slow performance

Client workstations were connected to different servers through a network. It was assumed that the client workstations weren't the ones responsible when a service was slow. It was more likely that the bottleneck[4] was the network or the servers used by the client workstations.

4.4.2 Pseudo packet count

The packet count in both models wasn't the standard packet count where one would count the number of packets going to e.g. port 25. It is only when tethereal is sure a packet really is a SMTP packet that it will label it SMTP. The packet count was therefore the same as the count of packets identified by tethereal to be e.g. NCP packets, as previously explained in section 3.5.1, page 18. Although it cannot be readily compared with other packet counts measured by other programs or in other ways, it can be compared with an observation which counts the packets in the same way, and this is what was done. The same way of analyzing the data with tethereal in the first part of the experiment, was also used in the second part of the experiment (again with tethereal), to see if the same amounts of traffic had been generated.

As for the two models, the exact packet count which formed the basis of the regeneration was not decisive, since the packet counts were only measurements relative to one another. The difference between the number of packets observed between interval A and B was going to be used, not so much interval A's packet count alone. Interval A's packet count on its own is correctly defined as the packets identified by tethereal to be of that service (e.g. IMAP packets).

4.4.3 Tracking one user

Tethereal was used to record the network activity from one real user. The problem was then how one could track one user throughout a day. The solution would be a

[4] the place making the service go slow

compromise. Assuming a typical user logs on a machine, does work, and logs out again, that machine was considered to be the equivalent of one user. A user may log in again to a different workstation, but the IP can still be regarded as one user. The use of different workstations is a phenomenon that is assumed to occur more frequently in educational organizations such as schools and universities where each user may follow several subjects, requiring him to move around to different rooms. It may also occur when there are not enough workstations for each user, so they need to share the available ones. The workstation is identified by its IP, and hence, tracing the actions originating from one IP roughly translates to the actions of one user.

4.4.4 Simulating multiple users

The same problem recurred when multiple user's worth of traffic was to be regenerated. The difficulty lay in how one virtual user related to one real user. A virtual user was implemented as it's own, independent process. This process could then, depending on the model, do service actions just as a normal user would. This way, multiple users could be simultated by running multiple virtual user processes.

A virtual user was close to one real user in the sense that either made decisions on what service to use in intervals of time. It was different in the sense that the virtual user would do very similar tasks at similar time intervals[5]. Only after observing how the virtual users behaved would there be evidence as to how this affected the simulation.

4.4.5 How to reproduce service usage

A way to reproduce traffic had to be devised. It wouldn't always work to reply old TCP sessions in order to regenerate server use. Passwords may have changed, and randomization in the form of nounces[6] or a change of the hash method used to scramble the passwords would make the authentication fail.

For instance, MS Windows file sharing with SMB uses encrypted passwords. SMTP may use SMTP–AUTH, a mechanism to enforce username and password for sending email. Since the authentication scheme may change, so can the scrambling of the username and password, rendering a replay of an old conversation useless.

While some simple services could be replayed this way, if a new service requiring a proprietary scrambling or encryption were to be tested, it wouldn't work.

As a general method, ripping off TCP and irrelevant IP headers and feeding the data to a program like netcat, or letting tcpreplay or flowreplay do it, could achieve the proper server response. For SMTP, DNS and HTTP, this ought to work, since they can be regarded as "dumb" services – using one such service doesn't affect using it again, and they don't really depend on interaction. They are fire and forget services, when a few optional details are disregarded[7]. They can in other terms be regarded as stateless. Sending an email or browsing a web page doesn't change anything the next time. You may do it over and over again and it would provide the same output. IMAP and file services are different. The output can be different each time, and they often require some sort of interaction. They can in this manner be looked at as stateful services.

An ill effect of recreating traffic this way would also be the possible disclosure of users' traffic. If a generic algorithm was constructed to extract random email

[5] The reason for this is explained in the following sections.

[6] a challenge–response object, new for each conversation, in practice making the username/password in encrypted form different for each time they are used.

[7] Such as cookies and session variables for HTTP, which are hacks to make HTTP a stateful service. The same commands can be sent to the SMTP server resulting in an email being sent.

or web conversations to be replayed later, sensitive data could be disclosed. Other eventualities could also happen: Say a user conversation with a web based email account is captured and replayed. This will introduce the risk that the email first sent by the user will be resent during the simulation, not just one time but many times. The same effect with a user doing business with a bank or flight reservation could lead to profound consequences. In reality, this would only occur at a small percentage of internet sites, since most of the serious ones use SSL and HTTPS, which is encrypted HTTP, making the traffic unable to be replayed. Also the use of session variables, cookies and passwords should protect against this, which essentially are the same as *replay attacks*.

A general way of regenerating the use of networked services would be to recreate them instead.

4.4.6 How to define the size of regeneration traffic

The only source of information for how large (in bytes) each service accommodated for, would be determined by the number of packets identified as that service. This would have limitations, since some services can send many packets with little actual data payload. This may be particularly true for services that need to keep track of what state they are in.

Both models relied on the number of packets observed as a measure of how much (bytes actual data) were transmitted, but this is for observational purposes only, since neither model can currently utilize the size for making more realistic traffic. A fixed size for each service action was chosen for simplicity.

4.4.7 How much granularity should there be

This was a matter of how small operations one could hope to simulate. If it was possible, it would be more realistic to be able to divide NCP/SMB traffic into this:

- SMB read
- SMB list directory
- SMB traverse directory
- SMB write
- SMB delete

This would require going down to a deeper level of traffic parsing, and add more development time and complexity to the project. This would not necessarily prove to be significantly more correct than a rougher granularity of simply combining these operations into one SMB action. It was the same with the rest of the services. A perceived average use of them was selected for implementation. SMB would read the equivalent of a MS Windows profile (about 1 MB), HTTP would download a web page with all its linked images to make it appear correctly, SMTP would send a plain text message of around 1 KB, and IMAP would download and delete all messages in the INBOX. For further details, look at the runXXXX.pl scripts.

4.4.8 Where to send service requests

When it came to SMTP and HTTP, another issue cropped up. To send emails with SMTP, one would need a recipient. As the author is not aware of any equivalent of /dev/null address for emails, and didn't want to send emails to innocent internet users nor be bothered to embark on some manually crafted mischievous mission to

set up X number of mailboxes spread out on the Internet, the solution lay bare. A trick often used to check if the email system works, is to send an email from your self to your self. This is also the approach taken here, in runsmtp.pl. This is not the same as would be a normal operation. The email server is relieved of the overhead in looking up the domains MX record in DNS, establishing a TCP session possibly over a WAN link for delivery, and to transmit the email. It could also happen that the domain is not found, and the mail would have to be bounced back. A possible "add–on" for the mail sending script would be to submit false emails to the server that would be certain to bounce. Another improvement would be to let the virtual users[8] mail each other, but this is again probably little different from sending email to oneself as long as there is only one mail server. In any case, it would never really replicate the normal use of an email server, since there is no distributed email addresses set up for this kind of testing known per date to the author.

4.4.9 What to trigger service use

A key point for the simulation was *when* service requests were going to be made, which was the same as *what* should trigger the use of services. Here lies the difference between the two models.

The first model aimed foremost to be able to recreate one hour of production time usage, from parsing a dump file for the same duration. In so doing, it was also going to tell something about the users of the system, in what way they behaved. The idea was to track users one by one, and observe the sequence of operations they made. This was implemented as a transition diagram, where each transition from one service to another was weighted with the number of such transitions seen for each user.

The first model was therefore represented as a graph:

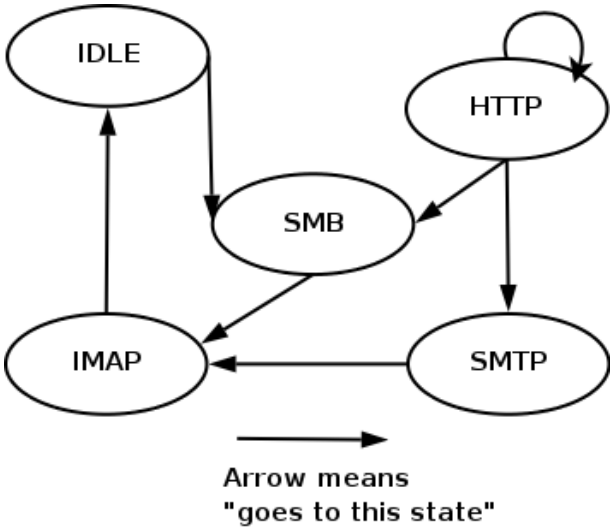

Figure 4.2: Model one, state machine, building a graph

This is a state machine, which is built up from the traffic seen on the network. State machines are commonly used for representing behavior and is often found in multi–agent systems and artificial intelligence[9].

[8] A pet name for the processes that runs the scripts that simulates the network service use / service events

[9] Further information about state machines can be found at

Each state has a pointer to all other states (including it self). Each time a service is used, the corresponding pointer's count is incremented.

The probability of using a particular service B coming from service A with the state matrix was calculated like this:

(Number of transitions from service A to service B) / (Total number of transitions from service A)

4.4.10 Idle users

In abstracting the network use into a state machine, the issue emerged as what to do with the users that were not logged on. It could be assumed that the amount of users was known, and from that number, it could be made out how many were active and in idle mode. Although this would seem preferable, it would mean that one would have to know the number of users before the simulation started. One precondition imposed over the project was to see how close it could get the real thing from just observing the network traffic passively. But for future work, it would probably be more accurate to have an option of telling how many users there are on the system.

To enable idle or logged off users, a new state was invented, labeled *IDLE*. If a user didn't make any requests within some predefined time, which is a parameter to model one, he would be considered to have entered an *IDLE* state, and the corresponding values in the state machine of model one would be increased. A small inaccuracy in the way this is implemented was incurred. Since the number of idle IPs wasn't known in advance when the simulation built up the state machine, but slowly increased, the result would be slightly less IDLE'ing for the virtual users than their real counterparts.

The *IDLE* state came thus in addition to the other ones that had been chosen earlier to be the most important and largest, see section 4.1 page 22.

For model two, solving the idle challenge had to be different. The idle state couldn't as easily be included, since the whole idea was to build it on a packet count for each service for each time interval. Thus, a list of time intervals with respective packet counts for each service was used, but without an *IDLE* service. To remedy this, the probability for doing one service action, for one particular time interval, was calculated this way:

(Number of packets in this interval for service A) / (number of packets of any service observed this interval + average of packets of any service higher than average number of packets seen for all intervals)

Assuming that nearly everyone is doing something at the peak of network traffic, the result is that the generator always will have a chance of selecting the IDLE state. On the other hand, in quiet intervals with small packet counts, the odds of triggering an event (service usage) are small. An effect of this is that the virtual users will be more active than their real counterparts when the eventlist services has more than average packets displayed, and less for the opposite.

Other ways of calculating the probability was also considered. Dividing by the average packet count resulted in too active virtual users, while dividing by the maximum number of packets seen in an interval yielded too lazy (idle) users. Dividing by the maximum value observed would be more true since one could then argue that every user was doing something at the time when the traffic load was at the highest. The maximum value could on the other hand also be a stray high value, which would impair the simulation. The chosen formula was deviced since a number between the average and the maximum packet count was needed.

http://en.wikipedia.org/wiki/State_machine

4.4.11 Parallelism

Both models assumed that one user can only be doing one thing at a time. This is arguably true for most occasions; few people really surf the internet while sending an email at exactly the same time. The same goes for reading email while surfing the web, but it may not hold for certain scenarios. Given that a user is working with his files, he may be downloading or uploading some changes to the network file server that takes some time, meanwhile he can be checking his email account. This was not directly modeled in either model – but indirectly nonetheless. As each virtual user is running in their own process, and spawns new processes to perform tasks such as check the email and download some files, the processes may not be executed right away. In a multi user computer system, the resources are limited, so e.g. the HTTP request may have to wait some time. This time can be enough for a new request to be generated in the meantime. Thus, the result would be that a file service operation executes simultaneously as a web service operation.

Another argument was that it scarcely mattered since, as was the premise of this thesis, the virtual users were not going to replicate the traffic as close to reality as possible, but on average. That left room for imperfect regenerations.

Adding to this was also the fact that the simulation supported more than one virtual user executing in parallel. The chance of service operations occurring in parallel would then increase exponentially as the number of virtual users increased. This was a substitute for situations where one user did several service operations at the same time.

4.4.12 Recreating traffic

The tools to recreate traffic are outlined in section 3.5.1, page 18. The challenge here was to find out how much traffic to recreate, when an event had been triggered by either model. It was decided to try to make small service requests rather than large ones, in the same way as one would redraw a time series using smaller intervals to increase the resolution. One such interval would then be the same as a small service operation.

Model one featured a way of telling how big the sessions were by counting the packets sent for each session, so an additional parameter was added to tell the packet count. If the packet count was larger than some predefined threshold, the request would be larger than default, and vice versa. This was however, not implemented.

4.5 Implementation of the generation part

An approach based on writing the least amount of code to generate the core functionality was used in implementing the models. This was necessary to be able to do it within the given time constraints. If required, improvements could be implemented were the need arose[10] There was thus a bit of trial and error involved.

In hindsight, to trial and error one's way through the coding and planning was a luxury that couldn't be afforded. When under the time constraints imposed on this project, it was learned that one must get it right the first time, and not experiment with developing different models or different approaches, since that wastes irreplaceable time.

This also led to the choice on speedy development language (Perl), tools and methods. The argument was that little could to be gained from picking the most difficult method if it only worked e.g. one percent better. It would be more

[10]This is somewhat like warfare strategy – you attack where the enemy is weak, you do not target his strengths head on. One starts to program where it is easy, where rapid progress is made.

efficient to choose and fully understand an easier method, which was quicker go develop and learn for others. A pitfall people and companies often fall into is trying to make the ultimate tool right away, ending up with spiraling development costs without a product to sell, still on the design board. More modern system development techniques counter this by *versioning* and development models like *eXtreme Programming* where small improvements are made continuously on a product.

4.5.1 Model one overview

Model one bases itself on the central data structure given in figure 4.3.

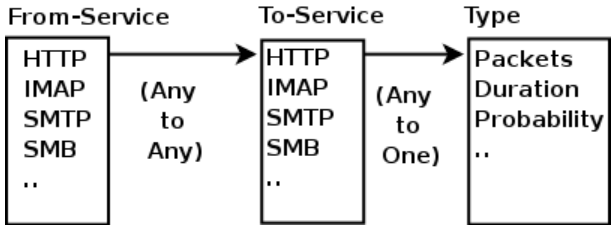

Figure 4.3: Model one's state array central data structure. This is a perl hash list (a data structure optimized lookup speed). To get the combined duration when a user goes from HTTP to NCP one would look up with Fromservice=HTTP, Toservice=NCP, Type=Duration, and get the sum of durations of NCP sessions coming from HTTP. This would then have to be divided by the probability for the same service sequence to get a duration roughly equal to one users NCP session duration after having used HTTP.

This data structure, the state array, stores the probability of a transition between each state. The probability is stored simply as a count of how many such transitions have occurred. In addition to the number of transitions, the duration of the session, and size of the session is also stored[11].

The data structure is generated from one or more libpcap dump files. They are parsed with tetheral, and tetheral's output is parsed for desired services. For instance, a stream of SMB packets to the CIFS/NetBios ports is classified as SMB packets by tethereal. The previous service from the sender of the SMB packets is used to increase the probability of going from the old service to the SMB service. Also, the duration of the old service is stored, and the packet count.

The data structure is written to a text file in the form of a matrix, one for each type of characteristic that can be used by model one (probability, size and duration). The data structure can then be parsed directly from the text file.

When model one is used to generate traffic, an interval parameter is used to control how often the script should check if a new service action should be made. The check is made by summarizing the probabilities and based on them, choosing a service to be used. A request for such an event including arguments is passed on for execution in a separate process. An example of a request for a service usage:

HTTP→ IMAP, size = 123, duration = 50.

This means that the virtual user should make use of IMAP for about 50 seconds and transmit 123 bytes of data. The size and duration parameter were not implemented for the IMAP, SMTP, SMB or HTTP service, so those actions will ignore those parameters, but duration is implemented in the IDLE state of model one.

[11] These are not implemented and used in the regeneration phase

After this, a new session will be chosen based on the probabilities given by the IMAP state.

4.5.2 Model two

Model two uses the central data structure given in figure 4.4:

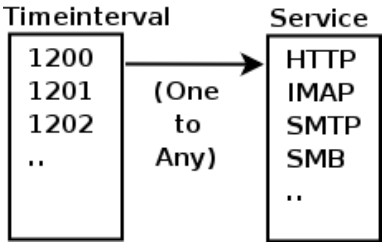

Figure 4.4: Model two's *eventarray* central data structure. This is also a Perl hash list. To get the probability for doing a SMTP event at 12:01, one would look up which interval 1201 is in Timeinterval, and SMTP in Service, and get a value. This value is used to calculate the probability of using SMTP in that time interval, as explained in section 4.4.10, page 29

.

Model two only sums up the packets seen in each time interval (typically two or more minutes). This is stored in an eventlist data structure (see figure 4.4).

The eventlist structure is also written to a text file in the form of a list. A time interval is the index to each service packet count. This text file can then be read into the eventlist.

When model two is used to generate traffic, an interval parameter is used to control how often the script should check if a new service usage should be made.

The main program then calculates the probability of a service event for the given time interval. The probability of doing one service is explained in section 4.4.10. A request for such an event is passed on for execution by a separate process.

4.6 The Perl scripts

The VSIM models are made up of Perl scripts which acts on a data structure which holds the probabilities for each possible action according to 1) the previous action (model one) or 2) given time (model two). The two models are started by executing vsim.pl (model one) or vsim2.pl (model two). All configurations and parameters are kept inside each script eliminating the need for command–line parameters. The files making up the simulation are:

The | means OR, so run[smb|http|imap|smtp].pl means there are 4 scripts; runsmb.pl, runhttp.pl, runimap.pl and finally runsmtp.pl. vsim[2].pl means vsim.pl and vsim2.pl. They are linked together with Perl *require* statements, and is entirely function oriented in that all functionality is expressed through functions. The script file names are also the names of the main function the scripts provide. The one exception is plcparser.pl and plcparser2.pl which main functions are parseplc() and parseplc2().

4.6.1 VSIM file structure

This is the files and directory structure of VSIM. Note that both models' files are here.

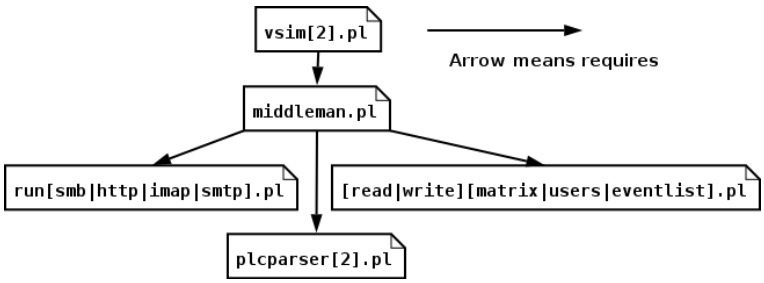

Figure 4.5: The Perl scripts. They are linked together like the arrows indicates, which is also gives the ordering of which scripts use which scripts.

```
drwxr--r--    2 lf            304 2005-05-06 12:32 data
-rwxr--r--    1 lf           2.3K 2005-04-26 11:34 middleman.pl
drwxr--r--    2 lf             48 2005-05-06 12:32 plcdir
-rwxr--r--    1 lf           4.6K 2005-04-22 19:21 plcparser2.pl
-rwxr--r--    1 lf           7.7K 2005-04-22 19:17 plcparser.pl
-rwxr--r--    1 lf           1.3K 2005-04-22 19:17 readeventlist.pl
-rwxr--r--    1 lf           1.1K 2005-04-22 19:17 readmatrix.pl
-rwxr--r--    1 lf           1.3K 2005-04-22 19:17 readusers.pl
drwxr--r--    2 lf            176 2005-05-06 12:32 run
-rwxr--r--    1 lf           7.5K 2005-05-05 14:18 vsim2.pl
-rwxr--r--    1 lf           5.7K 2005-05-05 14:17 vsim.pl
-rwxr--r--    1 lf           1.1K 2005-04-22 19:17 writeeventlist.pl
-rwxr--r--    1 lf           1.9K 2005-04-22 19:17 writematrix.pl
```

Used by model one – vsim.pl

plcparser.pl parses one or more libpcap dump files into a data structure used for state transitions – statearray.

readmatrix.pl reads a matrix configuration file (data/matrix.txt) into statearray.

writematrix.pl writes a matrix configuration file (data/matrix.txt) from statearray.

Used by model 2 – vsim2.pl

plcparser2.pl parses one or more libpcap dump files into a data structure containing service probabilities over time – eventarray.

writeeventlist.pl writes eventarray to an eventlist configuration file (data/eventlist).

readeventlist.pl reads an eventlist configuration file (data/eventlist) into the eventarray data structure.

Used by both model one and two

readusers.pl reads usernames, passwords and servers from data/users.txt into a data structure – usrinfo.

The data directory

```
-rwxr--r--    1 lf            201 2005-04-22 19:17 eventlist
-rwxr--r--    1 lf           1.3K 2005-04-22 19:17 matrix.txt
```

```
-rwxr--r--    1 lf           1.5K 2005-04-22 19:17 outmatrix.txt
-rwxr--r--    1 lf            578 2005-04-22 19:17 users.txt
```

data/eventlist is an editable list used to store the eventarray data structure.

data/matrix.txt is an editable matrix used to store the statearray data structure.

data/outmatrix.txt is an optional matrix generated from parsing dump files. It is thus interchangeable with data/matrix.txt.

data/users.txt is an editable list of user details read when the simulation starts.

The run directory

```
total 20K
-rwxr--r--    1 lf           1.9K 2005-04-22 19:17 runhttp.pl
-rwxr--r--    1 lf           3.7K 2005-04-22 19:17 runimap.pl
-rwxr--r--    1 lf           4.4K 2005-04-22 19:17 runsmb.pl
-rwxr--r--    1 lf           2.3K 2005-04-22 19:17 runsmtp.pl
```

run/runsmb.pl runs a SMB operation.

run/runsmtp.pl runs a SMTP operation.

run/runhttp.pl runs a HTTP operation.

run/runimap.pl runs an IMAP operation.

The plcdir directory is empty unless dumps are placed there. This can be the same dump files as in the plcdir used by capture.pl in the analyzer part.

4.6.2 vsim.pl

vsim.pl starts by calling startvsim(), which is the function that kicks off the simulation. It is mostly the same for model one and two except that each launches different models, modelone() and modeltwo(). Inside startvsim(), a lot of parameters are set up for the simulation:

Parameters

inputmatrix where the state matrix is.

inputusers where the user details list is.

outputmatrix where to save the state matrix.

plcdir where to look for dump files.

toptions options to pass on to tethereal for dumping traffic.

timeoutvalue seconds to timeout into idle state. When parsing a dump file, this determines how long it should take before an IP is considered to be in an IDLE state, after no traffic has been seen from it.

runtimesec seconds to run the simulation.

plist list of protocols we want to look for. These are matched against Tethereal's output.

tlist list of characteristics we want to look for. Three are currently implemented: Prob means the probability for a service to occur after each service, duration is the duration of that service and packets is the packet count of that service.

Functions

The main functions called are:

readusers() reads the user details list into a hash list.

readmatrix() reads the matrix into a hash list.

parseplc() parses one or more dump files into a hash list. This can be called instead of readmatrix()]

writematrix() writes the state hash list out to a matrix file. This can be called after parseplc().

modelone() launches simulation model one in one separate process, aka as one virtual user.

Readusers() reads in a list of users and user details like each user's mail server, home directory, password and more into a data structure called usrinfo. Then, readmatrix() reads probabilities and other variables into statearray. If no such matrix exists, one may choose to run parseplc() instead which parses one or more dump files into the statearray data structure. Then, writematrix() can be called to write the data structure to a file, which later can be read with readmatrix().

A sample state matrix, generated from dump taken on 29. April, 12-1300, IU/OUC:

duration:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	71839	1071801	9	2569	1834	82
IDLE	139168	1233664	312	15714	10126	101
IMAP	1	45	3230	321	7	1
NCP	3009	9797	161	155766	1439	1
SMB	3408	29078	22	1655	35065	1
SMTP	21	90	0	0	85	24

packets:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	355114	1609	23	4421	3758	5
IDLE	0	0	0	0	0	0
IMAP	22	4	5019	297	24	5
NCP	4427	220	301	5010967	777	3
SMB	3750	129	24	785	119403	3
SMTP	2	2	0	8	5	448

prob:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	1506	1609	23	4421	3758	5
IDLE	1540	21000	3	172	113	1
IMAP	22	4	52	297	24	5
NCP	4427	220	301	2620	777	3
SMB	3750	129	24	785	589	3
SMTP	2	2	0	8	5	3

It was decided not to normalize the data into percent values since that would loose accuracy. The services on the horizontal line are the "from" services, while the services on the vertical line are the "to" services. As an example, from IMAP to IDLE the prob (number of transitions) is 3.

The matrixes can be edited manually to facilitate different user behavior. For instance one can make the matrix so it would generate a circular use of services, stress only one, or make an arbitrary pattern - e.g by having HTTP always follow SMB usage. The matrix isn't size limited, and generic in the sense that it can be expanded to include other protocols, with no need for additional coding, other than in making a script to simulate the use of the service.

For each user in the usrinfo, a separate process is launched with modelone. So if there are 20 usernames in usrinfo/inputusers configuration file, 20 modelone functions are started as their own processes. Each process is one virtual user, and operates from his/her home directory found in the usrinfo data structure.

Modelone will run for runtimesec seconds, or up to runtimesec + durationsecs seconds. Durationsecs is how long each session is assumed to last. It may happen that a session is just started before runtimesec runs out.

For each interval, a check is made to see if the virtual user should use a service. This is based on what is stored in the statearray. This can be a HTTP request for instance. The service request is launched in yet another separate process. This is by design so that if the target server capacity is strained, it won't affect the ability to launch new requests.

4.6.3 vsim2.pl

Vsim2.pl starts by calling startvsim(), which differ from model one by using another data structure, see section 4.4. It has these parameters which are set up for the simulation:

starttime what time the simulation starts running in. One can give the present time, or fool the simulation into thinking it's e.g. 04:09 in the morning.

inputusersfile where the user details list is.

eventfile where the list of per time interval service traffic is.

plcdir where to look for dump files.

toptions options to pass on to tethereal for dumping traffic.

intervalsec seconds between each interval – setting this to one second will result in that for each second, a check is made to see if the virtual user should make use of any service.

runtimesec seconds to run the simulation.

packetthreshold if no more packets that this is found in an interval, the virtual user is doing nothing.

pseudosize default packets to send. It is needed to remain compatible with middleman/modelone function calls to runhttp/smb/imap/smtp. This model doesn't directly support the size of requests, nor is it implemented in the run scripts.

plist list of protocols we want to look for.

Functions

The main functions called are:

readusers() reads the user details list into the usrinfo data structure.

readeventlist() reads the event list into the eventarray data structure.

parseplc2() parses one or more dump files into the eventarray data structure. This can be called instead of readeventlist();

writeeventlist() writes the eventarray to an eventlist file. This can be called after parseplc2().

modeltwo() launches simulation model two for one virtual user.

Readusers() reads in a list of users and user details like each users mail server, home directory, password into a data structure called usrinfo. Then, readeventlist() reads time and probability values into eventarray. If no such file exists, one may choose to run parseplc2() instead which parses one or more dump files into the eventarray data structure. Then, writeeventlist() can be called to write the data structure to a file, which later can be read with readeventlist().

This approach is very much the same as model one, but instead of using states to simulate a user, it makes use of a time series from where probabilities for each traffic type are calculated. This model allows for the generation of the same traffic amounts at the same times of day. This builds on [3].

A sample eventlist, abbreviated:

Time	HTTP	IMAP	NCP	SMB	SMTP
10:40:52	0	0	55	28	0
10:40:54	34	0	191	4	0
10:40:56	660	0	948	46	0
10:40:58	3	0	650	12	0
10:41:00	0	0	3523	18	0
10:41:02	1	0	88	12	0
10:41:04	5	0	46	39	0
10:41:06	2	0	112	21	0
10:41:08	1	0	116	136	0
10:41:10	0	0	40	0	0
10:41:12	0	0	107	4	0
10:41:14	0	0	1784	8	0
10:41:16	0	0	1708	102	0
10:41:18	0	0	4443	8	0
10:41:20	0	0	7687	20	0

For each interval, a check is made to see if the virtual user should use a service. This is now based on what is stored in eventarray. This can be a HTTP request for instance. The service request is launched in yet another separate process. This is by design so that if the target server capacity is strained, it won't affect the ability to launch new requests.

The eventlist can be edited manually to facilitate different user behavior.

4.7 Deployment

To gather and observe network traffic, the equipment for the capture lab was put up. It featured a 2.4 GHz Pentium 4 processor with 512 MB Ram, one 80 and one 100 GB hard drive, one 100Mb Ethernet NIC for management and one 1Gb Ethernet NIC for packet capture. It was installed with Debian GNU/Linux Sarge operating system with Reiserfs filesystem and ICEWM window manager, and only left with SSHD running on the management network interface (100Mbit Intel e100).

The Gb Ethernet interface was plugged into the span port of the core switch which was handling the student and the employee network. The span port was

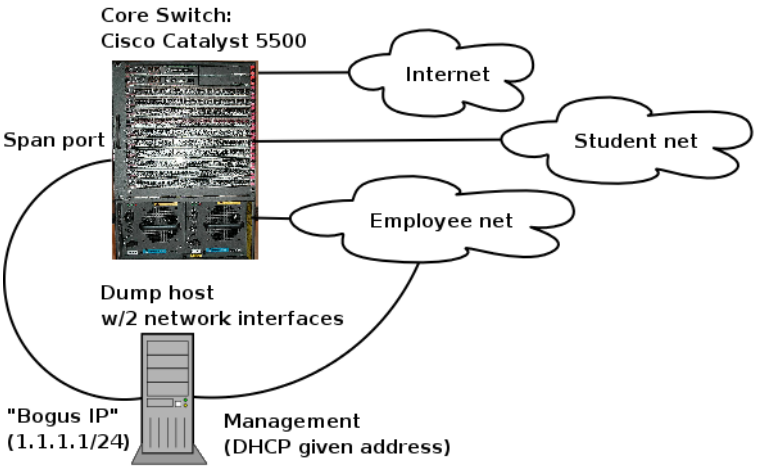

Figure 4.6: The capture setup

configured to receive all traffic to and from the student VLAN[12] (VLAN 30). The span port is a physical interface where the switch spews out all or selected traffic from one or more physical ports or VLANS. The core switch, a Cisco Catalyst 5500, was divided into several VLANs:

VLAN number	VLAN name	Network
30	student	128.39.74.0/23
20	ansatt	128.39.89.0/24
100	netlab	128.39.73.0/24

Table 4.1: The VLANs of the core switch

4.7.1 Preliminary observations from the student net

The *analyzer.pl* script was run to dump network traffic for several days. It generated graphs for this data which serve as the base for the simulation.

Due to increasing popularity from others to use the dump host to dump traffic from the span port, and time restriction, enough measurements were not taken to make out any standard deviance. To calculate a standard deviance would require observing the traffic at the same time of the day for several weeks. This is clearly too long when one considers the length of this thesis.

The graphs and observations are commented in section 5.1, page 42.

4.7.2 Test lab setup

To test the simulation, a virtual production system was built using 2 PCs connected with a crossover 100 Mbit Ethernet cable (TPX). The Server PC was a 500 mhz Pentium III with 512 meg ram and two 100Mbit NICs[13]. It was installed with Ubuntu Hoary, a patched version of Debian unstable (Sid). It was configured to have Reiserfs filesystem, OpenSSH SSH[14] server, Postfix SMTP server, Cyrus IMAP server, Samba SMB/CIFS server, and Thttpd HTTP server. All these services were available for the 10.0.0.0/24 network, which was only inhabited by the client.

[12] Virtual Local Area Networks, a way to put geographically separated LANS into one LAN
[13] Network Interface Cards
[14] Secure Shell Daemon, used to log in remotely

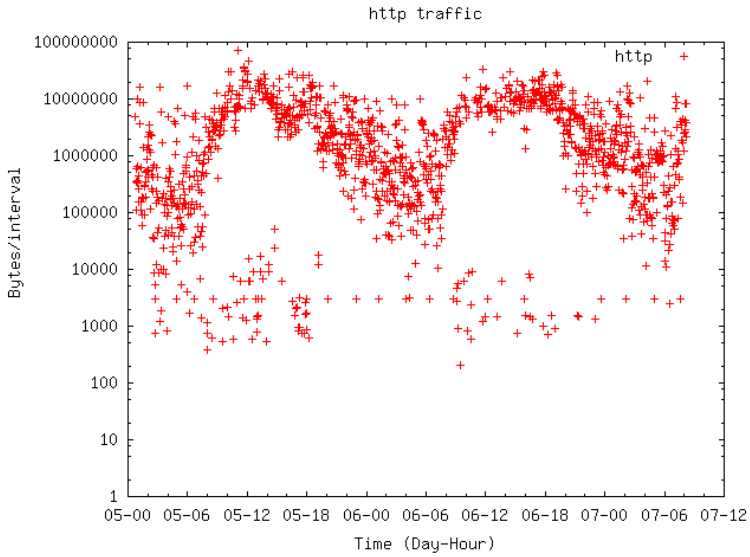

Figure 4.7: http traffic, 05-07 april

The client hosted the virtual users which were run against the server. It was configured in the same way as the server, but the only service left running was SSHD. The results are in section 5.2.

4.8 Analysis of the models

Both models were supposed to be able to simulate one user, in anticipation that it was going to be used to simulate additional users on the network. Originally, it was planned to dump from one IP only, and make a matrix and eventlist of that. But dumping from only one host would unveil that host's behavior, not the general behavior suitable for the simulation. It was therefore necessary to dump from more than one host in succession or in parallel. The latter was chosen due to the fact that it was faster. The resulting dump was then considered to be an average of the behavior of the users. The matrix and eventarray would first and foremost serve as a base to extract probability for events occurring, and to this purpose the sum of several hosts traffic was acceptable. What was not so clear was how to be able to generate traffic from only one user. That could be adjusted with how frequently each test for a new event is made in either model and how many virtual users were configured. Some experimentation was done to see out what matched best, but it needn't be the same for other systems. One virtual user is not exactly the same as one real user.

Model one was designed for one hour regeneration, so accusing it for not being able to simulate traffic for longer periods[15] would be beyond the point. It doesn't contain any variables related to how the probabilities vary over time. So when run, it just makes everyone (the virtual users) work equally hard all the time. It is important to recognize this also when the matrix is generated from dump files – it should not be fed with dump files that are dumped for longer than one hour. It should be fed with dump files from one hour, and that hour should be the one that are sought to be regenerated. This will ideally be the most productive hour.

[15]It would be like accusing a sponge for not being capable of cleaning the outside of the Taj Mahal.

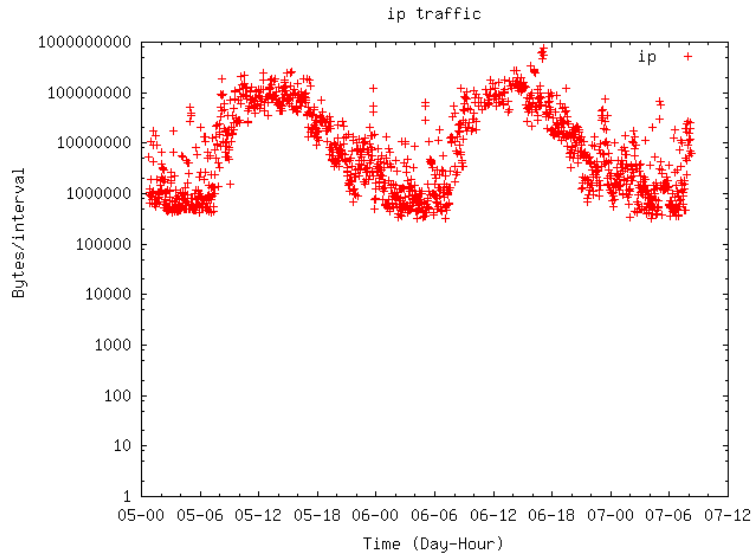

Figure 4.8: IP distribution, 05-07 april

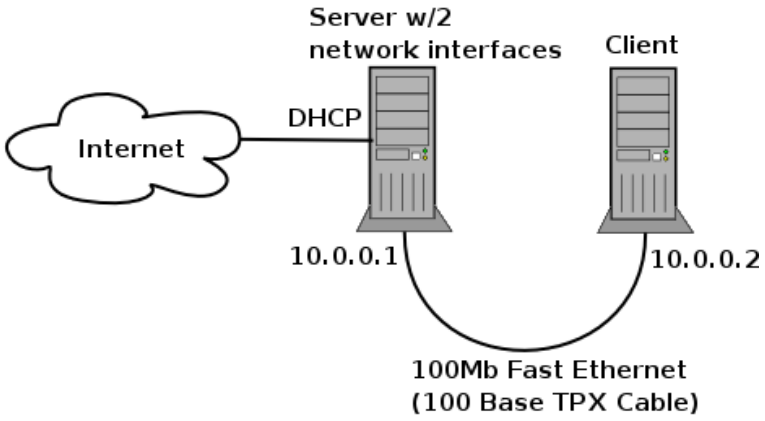

Figure 4.9: The VSIM lab

A major question was if the VSIM tool was able, on a real life system (production servers), to simulate usage correctly. Here lay the largest and perhaps most important uncertainty. Obvious shortcuts were made: Emails are sent to the same user who is sending them, thereby leaving the email server an easier job since it doesn't have to find out where to deliver the emails. Only the same HTTP pages are read over and over again, and if the server has some method of putting the filesystem in to memory, this will probably be done resulting in better performance than would be the case with normal requests which dance around the site. IMAP users would also not only download and delete their email; they would most certainly make folders, catalogue email, save copies of email, fill their quota and not be as well behaved as the virtual users. SMTP also only sends the same email over and over again, with no attachments which are very common.

Weighing up for a little of this was the fact that the virtual users would be slightly more active than their real counterparts as explained in section 4.4.10, page 29.

Another issue with both models were that they didn't simulate the fact that one

user may be doing multiple things at once - for instance having an IMAP session running at the same time he is surfing some web pages or fetching a file from the SMB server. The reason for this is in section 4.4.11, page 30. This can be mitigated by increasing the resolution of the events by shortening the time interval between each check for a new service to be carried out. Then a simultaneous IMAP and HTTP session would appear like this: Open IMAP. Browse web page. Browse web page. Read IMAP.

There's too much uncertainty to say that one virtual user is the same as a normal user based on the algorithms and approaches taken. The way to find out if they do match to a satisfactory degree was to do an experiment to observe if they looked similar – the virtual users were not in any way proven to be the equivalent of their human siblings, but by observation, they should appear similar as seen from the network.

4.9 Predictions and presumptions

Some predictions could be made as to how the system would look under the brunt of the virtual traffic generated for the setup described. As IU opens for students at 0700 and closes each week day at 2100, there should be a peak of traffic around the middle of the day when most students are in, and less and less the closer it gets to closing time. This should be reflected in the dump files which in turn should be reflected in the event list, which finally should protrude the observations from the regeneration lab.

An ominous concern had to do with the client host IDE/ATA hard drives. Since some of the run scripts are writing to disk, and IDE Hard drives are not in any way good at handling multiple requests for read/write, they are at risk of becoming a performance–limiting factor. A common consumer grade hard drive spinning at 7200 RPM will spin off anything between 10-25 MB/second – during sequential reads. For random accesses, the performance drops abruptly. That's why network servers tend to use parallel SCSI disks, which copes better with concurrent operations, and they also utilize other means to fight I/O contention like massive bucket–loads of memory to cache up the filesystem. Operations take much less time in memory as on disk.

This potential pitfall could be mitigated by using multiple client hosts, but that was put off for future efforts.

If the client harddrives wouldn't suffocate under the stress, the next potential bottleneck was the network itself. As described in section 3.2.1, page 8, 12 MB/second could theoretically be supported. If, on the other hand, there were competing programs waiting to use this resource of available bandwidth, it would degenerate since each connection has some overhead and there would be more and more such connections, effectively reducing the available bandwidth. If this happened, it would have an even larger chance of happening on a live system. A live system would lump in even more network latency and switching overhead to the frying pan.

In addition there was a concern that the CPU of the client or server would start to be stressed if too many processes fought for processing resources. Each virtual user has its own process, and spawns a new process for each service usage it tries to perform. It was clear that CPU contention could happen, as well as client memory shortage since each process requires its own memory address space to live in. Experimenting with large number of users would give some observational clues as to where this was going to hurt the most.

Chapter 5

Results

To run the simulation, the testbed described in section 4.7.2 was put up. Real standard UNIX user accounts were made for the virtual users, in addition to SAMBA accounts. Since this is exactly what a real user can expect, the simulation was very realistic. Adding users and configuring services required real system administration methods. There was thus no distinction between a virtual user's account and a real user's. This made it more realistic, but the price for this was increased overhead when e.g. adding users, which took time. This contributed to the fact that only a handful of experiments were executed before the allotted time ran out.

The main experiments were initially done with 10 virtual users[1]. It was also tested with 20 virtual users, but the result was that the client suffocated in less than two hours. The client program generated so many apparently stray processes that the client could not launch any more. After trying to limit the number of processes that were allowed to run, it ran out of available memory for the same reason. The exact reason for the stray processes may not have originated from the client – it was possible that the server, due to excessive and simultaneous load, wasn't giving the correct responses expected by the virtual user processes, making them timeout slowly instead of exiting quickly. This needs to be sorted out and fixed in future versions, but with 10 virtual users, the simulation seemed to go smoothly for several hours.

5.1 Observations from the IU/OUC student VLAN

Here the observations of the IU/OUC Student net as observed with the setup in section 4.7 are explained.

Figure 5.1 shows a plot of the core services from 24 hours, and if one imagines the end to start at the beginning, this looks similar to a wave. The waveform can be correlated to the working hours – a significant jump in activity around 0745 can be seen, which is peaking around 1300 and slowly descends into the evening.

A lot of NCP and HTTP traffic was observed, but little SMB, IMAP and SMTP. The small amounts of SMTP an IMAP was not predicted. But the reason might have been that in addition to regular MUA such as Mozilla Thunderbird, OUC has a web interface for reading/sending mail as well. OUC uses Horde, a similar web based email application as Squirrelmail. In addition, another web-based application which also has email capability called Classfronter is also used. The tiny amounts

[1] See section 4.4.4, page 26 for explanation.

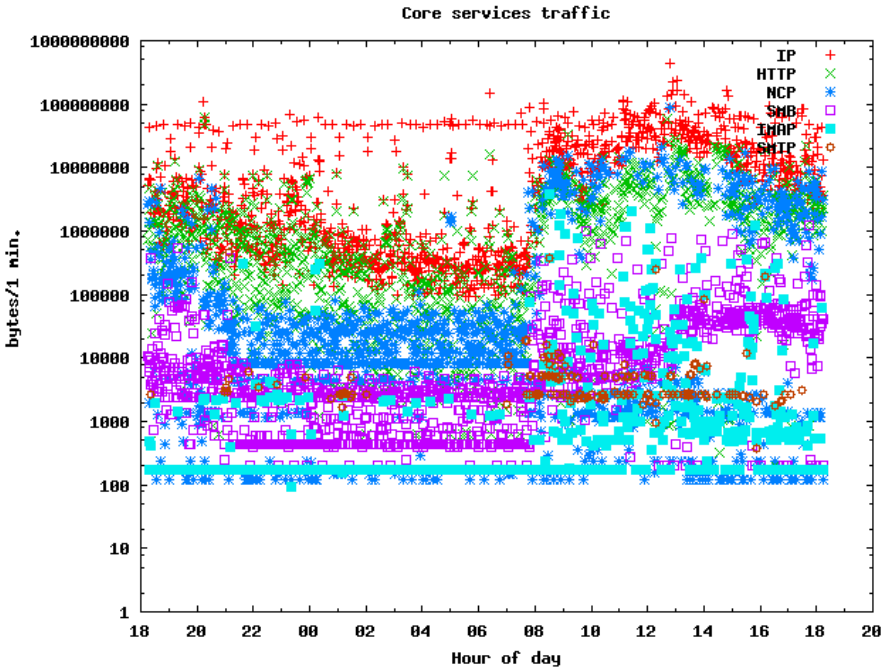

Figure 5.1: The distribution of the core services and IP, from 18:15 28 April to 18:15 29 April. The y–axis is scaled with log(10) increments. Note that HTTP and NCP combined looks similar to the total IP traffic seen, which means they stand for most of the traffic here. Gnuplot was used for plotting, but it didn't plot 0 values, perhaps since log 0 isn't defined. This made the IMAP and SMTP plots somewhat misleading since they consised of a lot of 0–observations. A count of different IPv4 addresses showed that 4520 addresses were involved. Investigation showed that 128.39.74.16 was a web server that external IPs connected to. That was a contributing factor to the large amounts of HTTP traffic.

of SMTP and IMAP may be explained by traffic going through HTTP instead. The web server which hosts the mail interface could either be on the same host as the IMAP/SMTP server, or the communication between them may be on a net that wasn't part of the net from where the dumping took place.

A script to check the total numbers of IP's seen were run, and reported 4520 different IPv4 addresses for this interval. Investigation showed there to be foreign addresses as well as local student net ones. This resulted in diluted data, but it was decided to carry on at this stage of the experiment, since it would not violate the goal of recreating the same characteristics.

Without using log(10) as y–axis scale, the same plot looked very noisy (figure 5.2.

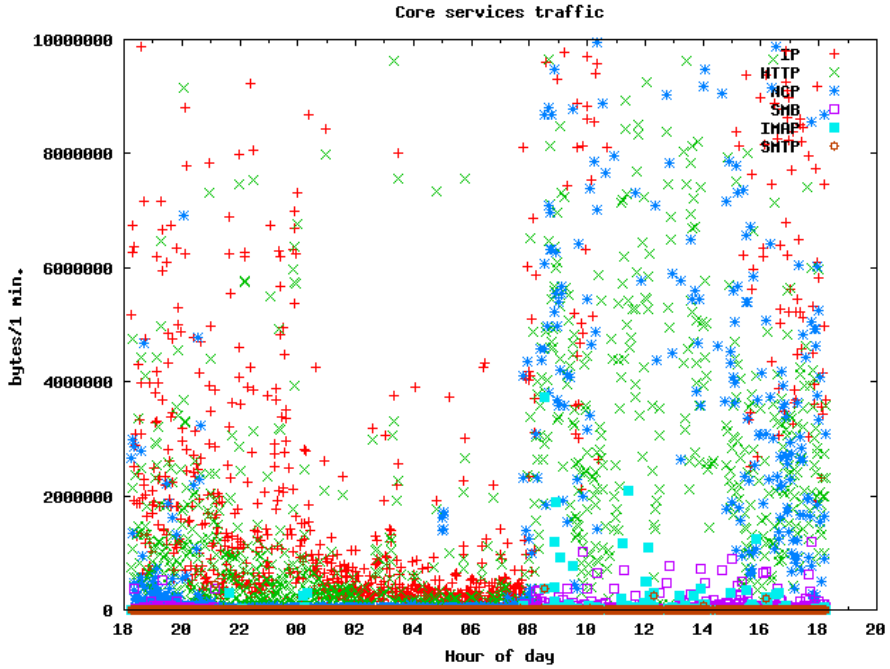

Figure 5.2: This is a plot of the same data as in figure 5.1; the distribution of core services and IP, from 18:15 28 April to 18:15 29 April.

Figure 5.3 is just a plot of the IP traffic of the same dataset, which gives a clearer view of the daily variation.

5.1.1 Observations from model one

This is a matrix generated from data collected from 12-1300 on April 29, 2005. The idle timeout was set to 60 seconds. As described in section 4.4.9, the prob matrix gives the number of transitions from the x–axis (horizontally listed) services to the y–axis (vertically listed) services, e.g. 5 transitions have occurred from SMTP to HTTP. The number of packets is the number of packets observed after coming from the x–axis service to the y–axis service, e.g. a total of 448 packets were observed of the SMTP service after coming from the SMTP service. The duration follows the same; it's the sum of the duration of the sessions for each service coming from each service, e.g. the sum duration when being in IDLE and coming from SMTP is 101.

`duration:`

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	71839	1071801	9	2569	1834	82
IDLE	139168	1233664	312	15714	10126	101
IMAP	1	45	3230	321	7	1
NCP	3009	9797	161	155766	1439	1
SMB	3408	29078	22	1655	35065	1
SMTP	21	90	0	0	85	24

packets:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	355114	1609	23	4421	3758	5
IDLE	0	0	0	0	0	0
IMAP	22	4	5019	297	24	5
NCP	4427	220	301	5010967	777	3
SMB	3750	129	24	785	119403	3
SMTP	2	2	0	8	5	448

prob:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	1506	1609	23	4421	3758	5
IDLE	1540	21000	3	172	113	1
IMAP	22	4	52	297	24	5
NCP	4427	220	301	2620	777	3
SMB	3750	129	24	785	589	3
SMTP	2	2	0	8	5	3

The large amount of HTTP can be explained by the fact that the student net covers a web server. This allows for external IPs to affect the measurements, which has happened here. The external web browsing results in much HTTP and many HTTP to IDLE transitions since it is plausible they leave the site or don't use any other services on the student net. When they are put into IDLE, after hearing no more of them for *timeoutvalue* seconds, they are timed out to go from IDLE to IDLE again. That is a plausible explanation as to why there are so many IDLE to IDLE transitions and long duration for the same transition. It was decided to carry on regardless[2].

There seemed to be few circular service requests where several services were used in succession. It looked as if HTTP, SMB/NCP was used for the most of the time, and use of other services was used only now and then, followed by HTTP and SMB/NCP. The abundance of SMB to SMB transitions may be partly explained by the fact that SMB/NCP is a stateful service requiring status information to be sent even if no actual file transfer is done.

The large amounts of HTTP to SMB/NCP transitions may be because of interdependencies – a HTTP action could lead to NCP usage. This may occur when the client browser is configured to put temporary files on a network file server. This would generate a NCP/SMB action. Then, the opposite would also be true, seeing many SMB/NCP to HTTP connections.

5.1.2 Observations from model two

This is a part of an eventlist from data collected on April 29 2005, with an interval of two seconds. The list displays number of packets in each interval as defined in section 4.4.10.

[2]The external traffic could be filtered out by adjusting parameters in the scripts if desired.

Time	HTTP	IMAP	NCP	SMB	SMTP
10:06:21	244	0	40	148	0
10:06:23	241	0	3628	10	0
10:06:25	0	0	63	2	0
10:06:27	3	0	112	6	0
10:06:29	0	3	5008	0	0
10:06:31	0	0	2960	28	0
10:06:33	2975	0	2166	32	0
10:06:35	1861	0	3576	8	0
10:06:37	307	0	412	108	0
10:06:39	182	0	1299	22	0
10:06:41	364	0	3949	38	0
10:06:43	8	0	302	0	0
10:06:45	9	0	2348	0	0
10:06:47	30	0	2967	26	0
10:06:49	91	0	4304	2	0
10:06:51	57	0	720	2	0
10:06:53	0	0	5515	0	0

This eventlist shows the majority of packets being NCP, followed by HTTP and SMB, and minor amounts of IMAP and SMTP. This corresponds to observation of the matrix and graphs of the traffic distribution.

5.1.3 Uncertainty

To give a clue as towards how certain these results were, observations from the same timespan from the matrix and the eventlist were compared. This told how much each models way of counting packets differed. Ideally, the numbers should be equal since they both used practically the same algorithm:

- The sum of HTTP packets from the event series interval 12-1300 is 355655 packets. The sum of HTTP packets from the matrix from 12-1300 is 364930, which is 9275 packets more, or 2.61%, in favor of the matrix.
- For NCP, the eventlist gives 5032429 packets and the matrix gives 5016694, which gives 15735, or 0.31% more packets in favor of the timeseries.
- For IMAP both observations give 5371 packets and 0.00% difference.
- For SMTP, both give 465 packets, equal to 0.00% difference.

It looked as if there were no apparent connection between the two models packet counts, since both more and less packet numbers were observed for each model. Sooner, it can be regarded as error rates which appear to grow as the number of packets grow for at least HTTP and NCP. The reason for this is not clear, but as the errors were relatively small, they were ignored.

Another way to describe the correctness of the matrix was to count each inbound and outbound transition to see if they made sense:

- Inbound to HTTP, from all other services, there is a total of 9816 transitions. Outbound to all other services, there are 9741 transitions.
- Inbound to IDLE there are 1829 transitions, outbound 1964.
- Inbound to IMAP, there is 352 transitions, outbound 351.
- Inbound to NCP and SMB, there is 8857 transitions, outbound 8798.

- Inbound to SMTP there is 17 transitions, outbound 17.

The fact that these numbers didn't match for all services, can be explained by the fact that at some time the dump was ended, leaving some IP/Users in each state. This would result in more inbound than outbound transitions. The opposite is not true since all users are regarded as in IDLE state before they do anything else. The exception is the IDLE state since that's where all IP/users start, allowing for more outbound transitions since not everyone is expected to return to IDLE state.

Services that were not much used had fewer transitions, like IMAP and SMTP. These had a better chance of getting an equal number of inbound and outbound transitions, since there was a higher chance of not interrupting any ongoing IMAP or SMTP sessions when stopping the dump.

The difference in inbound and outbound therefore seemed to increase with the number of transitions. But for using the matrix for regenerating traffic, the differences were tiny and produced little effect on the probabilities of entering each state.

A different source of uncertainty was the nature of the experiment. Since it was done with real servers and processes, there was always the chance of something going wrong. It was not an ideal enviroment[3], and this came to show when the client host's memory was depleted during the testing of model two's 24 hour simulation. Another time, the THTTPD daemon providing the HTTP service on the server died. These are problems that need to be adressed in future versions.

A possible uncertainty in the observation was the packet loss rate when dumping. A big burst of packets could have been too much for tethereal to deal with. No packet loss were seen when the methods were tested, so it was unlikely that any significant number of packets were lost when the real dumps used for simulation were made. To be fair, tethereal was only a part of a long chain of factors that could have attributed to packet loss. The switch with the span port could have difficulties in processing all the packets; the capacity of the cabling from the switch to the dump host could be insufficient, as could the Ethernet card or the hard drive used to write the dump files.

Another uncertainty connected to the packet dumping were if the dump wasn't taken during a normal day. For instance, a virus or worm attack, a deadline or something else might have affected the data that was assumed to be representative for a normal day. It would lead to different results. To get more correct results, more measurements and dumping would be needed. Only after collecting data for a longer period could the statistical properties be calculated to more correctly describe the distribution of a normal, average day.

5.2 Results of model one

These are the results of the simulation in the configuration described in section 4.7.2. The methodology used to test the simulation for model one was this:

1. Make a matrix from one hour dump files taken from the IU/OUC LAN.
2. Make a plot of the distribution of the same time interval from the dump files.
3. Run a simulation using that matrix in the vsim lab.
4. At the same time, dump the traffic that is generated this way.
5. Make a plot of the distribution of the new dump.

[3] as in an enviroment rid of threats, obstacles, competition and things that can go wrong.

6. Make a new matrix of the same dump.
7. Compare the plots and matrix from the simulation lab against the plots and matrix from IU/OUC.

5.2.1 Matrix from one hour dump, taken on April 29, 12-1300

The matrix used as input for the next simulation using model one was the same as in section 5.1.1, page 44. The matrix correlated to the graph containing the same time interval.

5.2.2 Plot from one hour dump from the IU/OUC student VLAN

The distribution of core services at IU/OUC, from 12-1300 29 April, is presented in figure 5.4.

5.2.3 Matrix from one hour simulation dump

This is the matrix generated from the simulated traffic seen in Figure 5.4. Ten virtual users were configured, and the simulation ran for one hour.

duration:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	5117	0	2	0	2161	0
IDLE	0	0	0	0	0	0
IMAP	0	0	1	0	73	0
NCP	0	0	0	0	0	0
SMB	1446	0	2	0	1175	0
SMTP	0	0	0	0	0	0

packets:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	17726	3	2	0	2989	0
IDLE	0	0	0	0	0	0
IMAP	0	0	73	0	27	0
NCP	0	0	0	0	0	0
SMB	2993	0	25	0	66750	0
SMTP	0	0	0	0	0	0

prob:

	HTTP	IDLE	IMAP	NCP	SMB	SMTP
HTTP	79	3	2	0	2989	0
IDLE	0	3	0	0	0	0
IMAP	0	0	0	0	27	0
NCP	0	0	0	0	0	0
SMB	2993	0	25	0	34	0
SMTP	0	0	0	0	0	0

Note that due to the fact that the clients are connecting from only one IP, the prob and timeout values cannot readily be compared with the original matrix.

By dividing packet numbers with those of the original matrix, a comparison was made to see if the simulation had been consistent in making the same amounts of services:

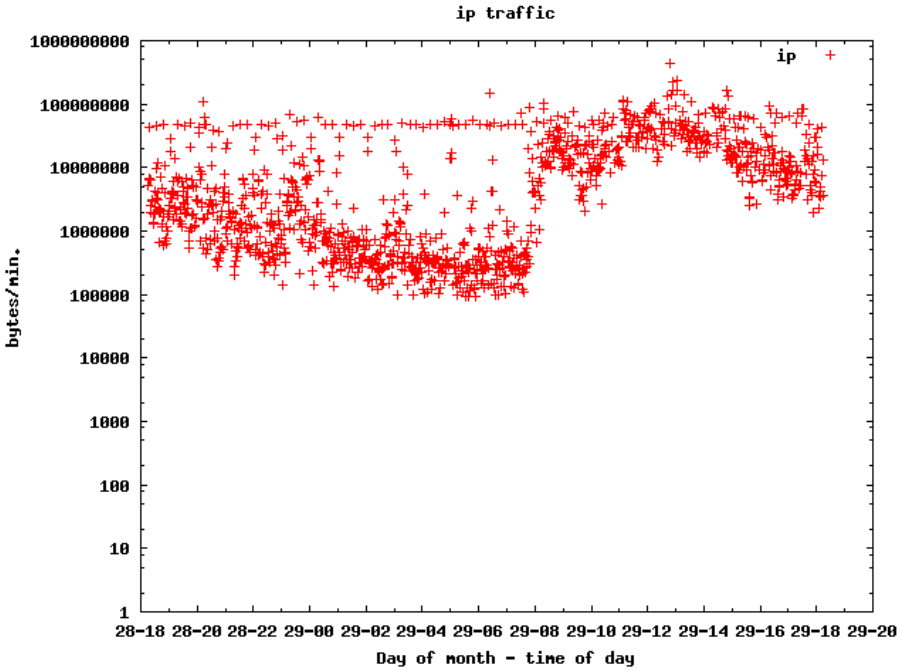

Figure 5.3: The distribution of IP, from 18:15 28 April to 18:15 29 April. The apparent straight lines at 100 000 000 bytes per 2 minute may appear because of some automatic non–human generated IP traffic appearing in regular intervals.

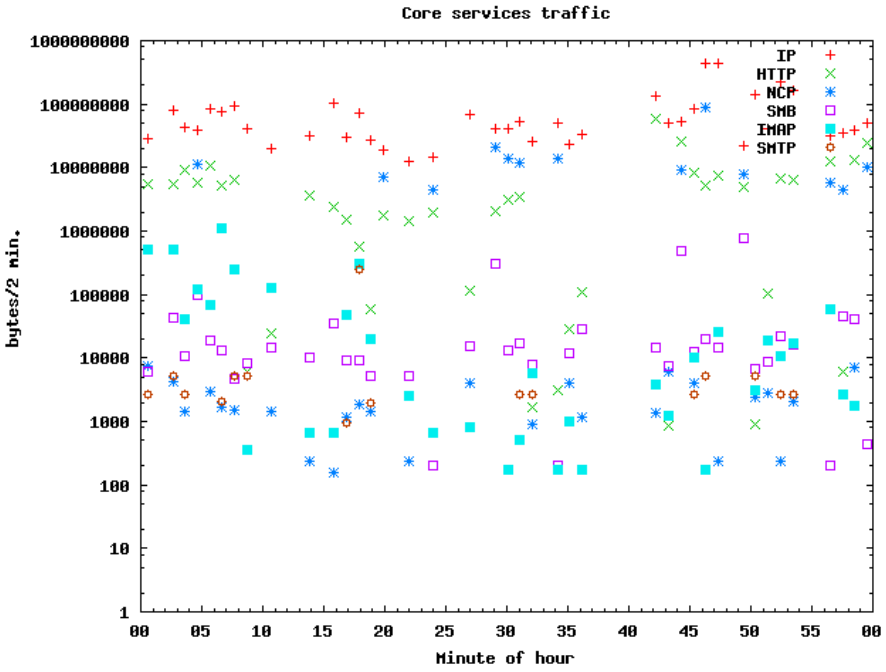

Figure 5.4: The distribution of core services at IU/OUC, from 12-1300 29 April. Note that HTTP and NCP combined look similar to the total IP traffic seen, which means they stand for most of the traffic here.

- HTTP to HTTP = 17726 / 355114 = 0.0499
- HTTP to SMB = 2993 / (HTTP to NCP + HTTP to SMB = 4427 + 3750) = 2993 / 8177 = 0.366
- SMB to HTTP = 2989 / (NCP to HTTP + SMB to HTTP = 4421 + 3758) = 0.365
- SMB to SMB = 66750 / (NCP to SMB + NCP to NCP + SMB to NCP + SMB to SMB = 785 + 5010967 + 777 + 119403) = 66750 / 5131932 = 0.013
- SMB to IMAP = 73 / (NCP to IMAP + SMB to IMAP = 297 + 24) = 73 / 321 = 0.227

Ideally, these numbers should be equal to one another, giving consistent differences, but they were not. This may be since this was only one test, and the model didn't guarantee the exact same results since it incorporated *probabilities* for using each service.

The inbound and outbound transitions can also be compared as previously explained in section 5.1.3:

- Inbound to HTTP, from all other services, there is a total of 2994 transitions. Outbound to all other services, there are 2993 transitions.
- Inbound to IDLE there are 0 transitions, outbound 0.
- Inbound to IMAP, there are 27 transitions, outbound 27.
- Inbound to NCP and SMB, there are 3018 transitions, outbound 3016.
- Inbound to SMTP there are 0 transitions, outbound 0.

The 0 observations of the IDLE state has got to do with the *intervalsec* variable, which controls how often a check should be made to run a service action. If it is set to less than the *timeout* variable, controlling how long time it should take for an IP/user to be considered in IDLE state when no service requests are observed from him, the probability of entering IDLE state is small.

Otherwise, the same reasons as in section 5.1.3 applies here.

5.2.4 Plot from one hour simulation dump

This is traffic distribution generated from the simulated traffic made by model one with the input matrix in section 5.1.1. Ten virtual users were in action for one hour, producing this traffic distribution (figure 5.5). The traffic looks much smoother here, and much of the noise seen in figure 5.4 is eliminated. The volume is roughly around 1 KB per 2 minutes, whereas originally 100 MB per 2 minutes. This is mainly due to using predefined service actions with fixed sizes. The simulation sends to few bytes per service action. This is an area that could be improved. Another factor is that only 10 virtual users were deployed.

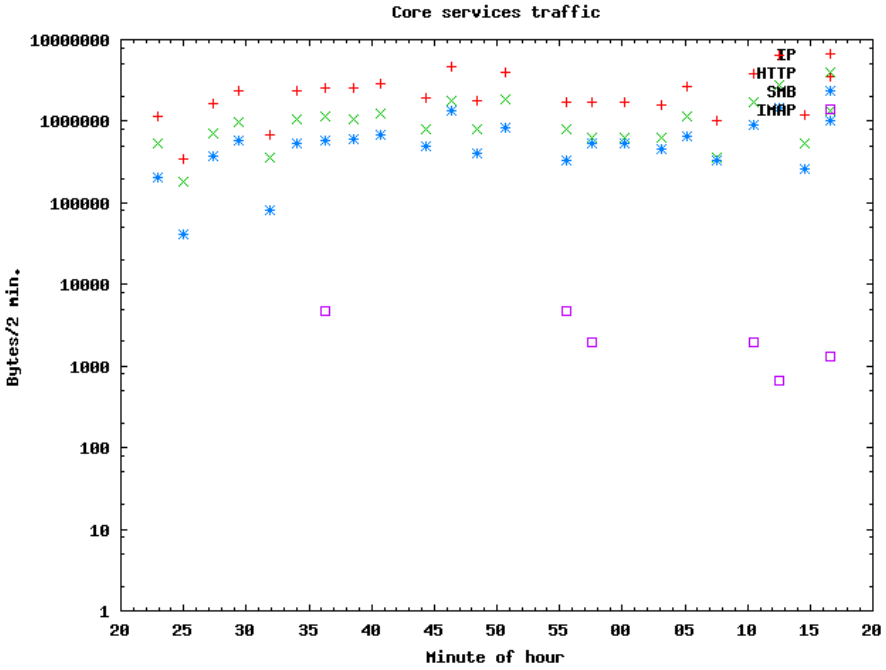

Figure 5.5: The distribution of core services as seen from the simulation with 10 virtual users. Note that there was no SMTP traffic here since none was observed.

5.2.5 Results with 20 virtual users

Another simulation was run with 20 virtual users, with the same input matrix. The result is plotted in figure 5.6.

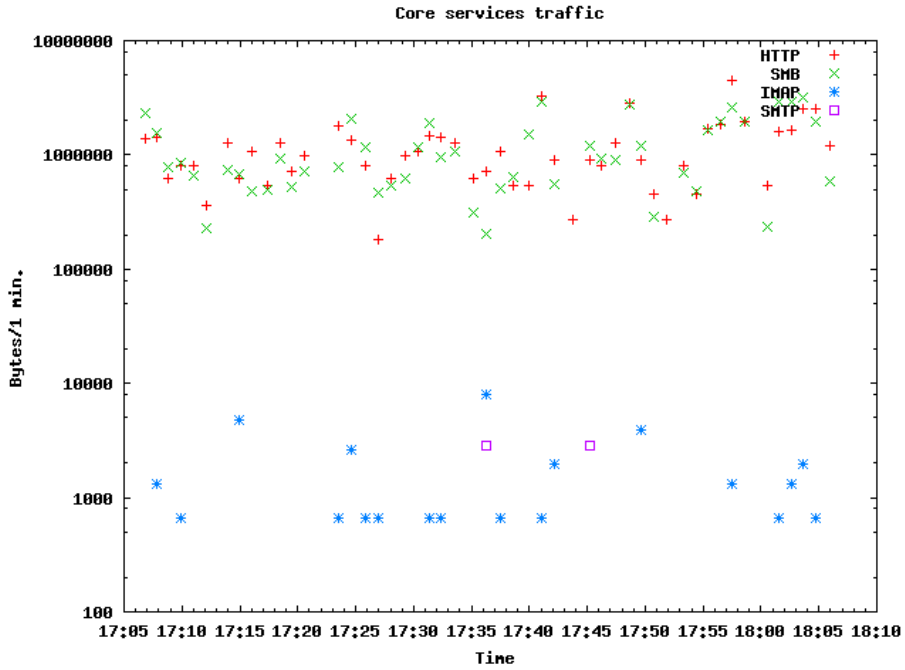

Figure 5.6: The distribution of core services as seen from the simulation with 20 virtual users. The main difference from the one with 10 virtual users are the inclusion of more SMTP and IMAP. Note that this is per one minute interval when comparing with figure 5.5 which is per 2 minute interval. The measurements of the data actually ends up at twice the rate. This is due to the doubling of virtual users.

The results of model one were promising. Increasing the number of virtual users increased the similarities between the results. It looked more like the original observations. On the other hand, the traffic seemed a tad more coherent and with smaller bursts.

The matrix generated from the simulation showed that the doubling of virtual users more than doubled the traffic. The larger the services, the more they increased. In the 20 virtual user simulation, IMAP to IMAP gave 2.1 times more packets than the 10 virtual user simulation. HTTP to HTTP gave 2.6 times more packets, and for SMB to SMB, the increase was 3.7. The probabilities showed roughly the same trend.

```
duration:
          HTTP      IDLE      IMAP      SMB       SMTP
HTTP      5203      0         23        1870      0
IDLE      0         0         0         0         0
IMAP      14        0         19        80        16
SMB       1858      0         16        1716      2
SMTP      0         0         0         4         0

packets:
          HTTP      IDLE      IMAP      SMB       SMTP
```

```
HTTP      46493     3         4         6638      0
IDLE      0         0         0         0         0
IMAP      8         0         155       78        2
SMB       6634      0         82        246472    4
SMTP      0         0         2         4         24

prob:
          HTTP      IDLE      IMAP      SMB       SMTP
HTTP      92        3         4         6638      0
IDLE      0         3         0         0         0
IMAP      8         0         2         78        2
SMB       6634      0         82        53        4
SMTP      0         0         2         4         0
```

A method to find an uncertain estimate of how many virtual users the interval of original traffic corresponded to can be made by dividing numbers from the original matrix from the IU/OUC VLAN by the equivalent numbers of the 10 and 20 virtual user simulations: If the HTTP to SMB observation is used, divided by the 10 virtual user simulation, the interval corresponds to this:

$$\frac{(\mathrm{NCP}+\mathrm{SMB})}{x} = \frac{(4427+3750)}{x} = \frac{2993}{10}$$

where X is number of virtual users, which when solved gives X = 27.3 virtual users.

Using the simulation with 20 virtual users, the interval corresponds to

$$\frac{(\mathrm{NCP}+\mathrm{SMB})}{x} = \frac{(4427+3750)}{x} = \frac{6634}{20}$$

X = 24.61 virtual users. This is not the same as with 10 virtual users, so this estimate is quite uncertain. The estimation tells how many virtual users are needed to generate the same number of *transitions* as the real users made. The same could be done with the other characteristics. By experimenting with the size and duration of the requests, and following this method, roughly the same number could be reached for prob, duration and size. It would then be reason to say how many virtual users corresponded to the traffic seen in an interval. If the number of real users in this interval was found by e.g. counting the number of IPs involved, an estimate of how many real users a virtual user corresponded to could be made.

5.3 Results of model two

These are the results of the simulation in the configuration described in section 4.7.2. The methodology used to test the simulation for model two was this:

1. Make an eventlist from 24 hour dump files taken from the IU/OUC LAN.
2. Make a plot of the distribution of the same time interval from the dump files.
3. Run a simulation using that eventlist in the VSIM lab.
4. At the same time, dump the traffic that is generated this way.
5. Make a plot of the distribution of the new dump.
6. Make a new eventlist of the same dump.
7. Compare the plots and eventlist from the simulation lab against the plots and eventlist from IU/OUC.

5.3.1 Eventlist from 24 hour dump

The eventlist from 28-29th of April was 3600*24 lines long as the interval was set to one minute, so only a part of it is included here. Increasing the interval would have given more packets for the assorted services, but decreased the resolution of the variations – if a plot were made, it would be more likely to have jagged curvature as opposed to smooth curves with fine resolution. An example of the latter is the eventlist with 2 second interval given in section 5.1.2.

Time	HTTP	IMAP	NCP	SMB	SMTP
07:30:08	60	0	95	4	0
07:31:21	113	2	62	16	0
07:32:31	69	0	64	6	0
07:33:39	14	2	234	12	0
07:34:47	122	0	254	8	0
07:35:49	20	2	92	16	0
07:37:01	11	0	66	4	0
07:38:03	24	2	138	18	0
07:39:07	317	2	152	4	0
07:40:08	20	0	94	4	0
07:41:21	24	2	62	4	0
07:42:34	70	0	65	18	0
07:43:46	14	2	1430	16	0
07:44:47	91	0	118	4	0
07:45:49	47	2	92	16	0
07:46:52	79	0	38219	4	0
07:47:53	5	2	13970	28	0
07:48:54	546	90	7601	182	0
07:49:59	300	4	150	298	0
07:51:01	250	0	1070	62	0
07:52:04	262	2	395	288	0
07:53:05	95	2	458	70	0
07:54:06	94	0	19001	8	0
07:55:07	1764	2	21874	188	0
07:56:08	250	0	18679	20	0
07:57:09	1116	2	13231	32	0
07:58:10	542	0	11209	32	0
07:59:11	1175	8	2094	36	0
08:00:13	540	0	2572	52	0
08:01:14	988	2	2833	22	0
08:02:15	4750	0	3159	182	0
08:03:16	2502	2	1156	20	0
08:04:17	142	100	621	134	0
08:05:18	52	2	628	354	0
08:06:19	401	0	16743	68	0
08:07:20	33	2	28928	222	0
08:08:21	3354	0	6645	138	0
08:09:24	2870	8	1965	112	0
08:10:25	735	0	733	2304	0
08:11:26	1955	2	1305	846	0
08:12:27	1500	0	1852	164	0
08:13:28	2980	2	40336	1078	0
08:14:29	1095	0	9061	1200	0
08:15:30	3505	2	26550	737	0
08:16:31	2504	0	17337	134	0

08:17:32	3632	2	65464	881	0
08:18:33	1537	6	70744	1195	0
08:19:34	1184	39	74283	2085	0
08:20:35	2642	0	46675	1377	0

The eventlist showed little IMAP and SMTP which correlated to the graph containing the same interval. This eventlist was used as input for the next simulation using model two.

5.3.2 Plot from 24 hour IU/OUC dump

The plot of the 24 hour dump between 28-29 April is given in figure 5.1.

5.3.3 Eventlist from 24 hour simulation dump

The 24 hour simulation, vsim2, was launched with 10 users and 5 seconds interval between each check for new possible service action. As noted, there were problems running the simulation for an extended period of time, since the client host was silently strangled by virtual user processes gone astray. But until suffocation, and with 10 users, it ran well for some hours.

The results of these hours seemed to share similarities with the plot. More specifically, the acceleration of service requests from 0740 and onwards looks to be depicted here as well. Note that the simulation was configured to start at 0740, so the time values here are off by 8 hours and 10 minutes:

Time	HTTP	IDLE	IMAP	SMB	SMTP
15:52:46	0	0	0	216	0
15:57:22	0	0	0	216	0
15:58:35	80	0	0	0	0
16:00:18	0	0	0	3672	0
16:01:27	0	0	0	2808	0
16:02:47	80	0	0	864	0
16:04:12	80	0	0	0	0
16:06:58	0	0	0	432	0
16:08:29	0	0	0	2376	0
16:09:39	0	0	0	2592	0
16:10:48	0	0	0	2160	0
16:11:57	0	0	0	3888	0
16:12:59	80	0	0	1944	0
16:14:56	0	0	0	1080	0
16:15:58	80	0	0	433	0
16:17:10	160	0	0	648	0
16:18:12	240	0	0	864	0
16:20:27	0	0	0	432	0
16:21:38	0	0	0	2989	0
16:22:39	0	0	0	3059	0
16:23:45	320	0	0	1728	0
16:24:52	0	0	0	1080	0
16:25:57	160	0	0	432	0
16:27:18	320	0	0	216	0
16:28:32	160	0	0	4104	0
16:29:33	0	0	0	2592	0
16:30:34	400	0	0	3456	0

5.3.4 Plot from 24 hour simulation dump

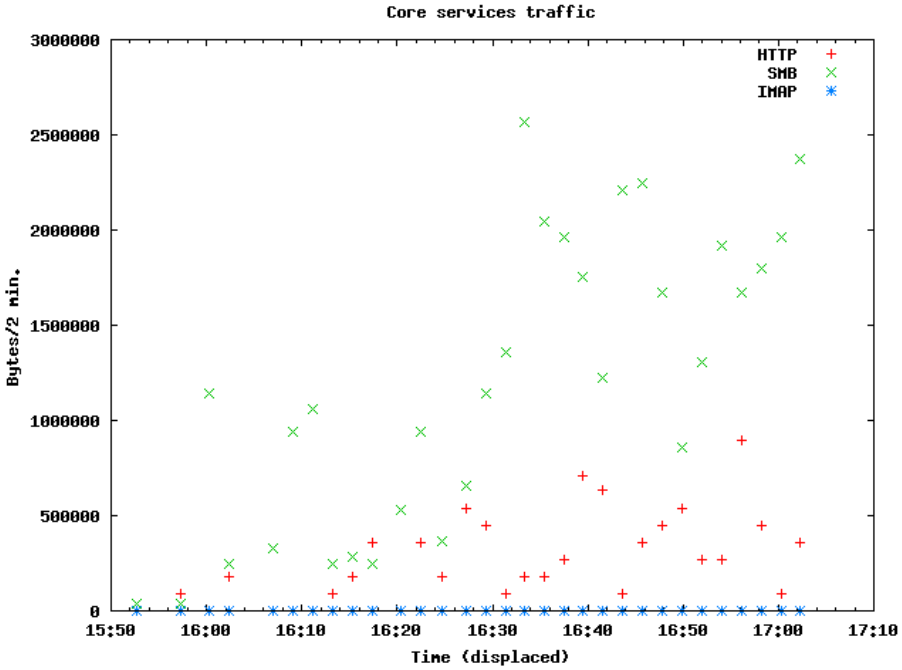

Figure 5.7: The distribution of core services from the VSIM lab, model two.

In figure 5.7, note that the simulation was not started in real time, so the x–axis timescale is not correct. The simulation was configured to start at 0740, so the time values here are off by 8 hours and 10 minutes. This was done since that time interval saw the greatest change of traffic volume. It was also not run for more than 2 hours and 20 minutes. The distinctive sudden leap of service usage seen in figure 5.1 can also be seen here, although much smaller due to the fact that only ten virtual users were deployed and the increased x–axis resolution of the plot. The simulation was run with *intervalsec* set to ten seconds, with ten users. Due to the absence of SMTP, it was not plotted.

Plotting the same graph with a log plot yielded the plot in figure 5.8. The same increase in HTTP, SMB and also IMAP can be seen. The values are very small compared to the original traffic volume this time series is based on, but that is because only ten users were deployed. If more were put into action, the curve would probably grow in volume, in the y–axis direction, and be denser.

The important factor was that it appeared to have similar traits as the original observation with regards to curvature. Another important element making it look more averaged, less bursty and less curvy than the original observation, is the way the IDLE state is calculated when probabilities for making each service are done in model two. The operation involves dividing by a number[4] which results in a varying threshold in the graph where time intervals with values beneath are more likely to fall in IDLE state than those with values above which are more likely to result in doing a service action.

This helped to even out the curves to some degree, and is evident in figure 5.9. It also looked as if the simulation became exhausted and not so virile as time went on, making less and less service actions. This was due to the unintended depletion

[4] see section 4.4.10

of client resources as described in section 5. The horizontal observations looking like lines are likely to be due to the fixed size of the service actions.

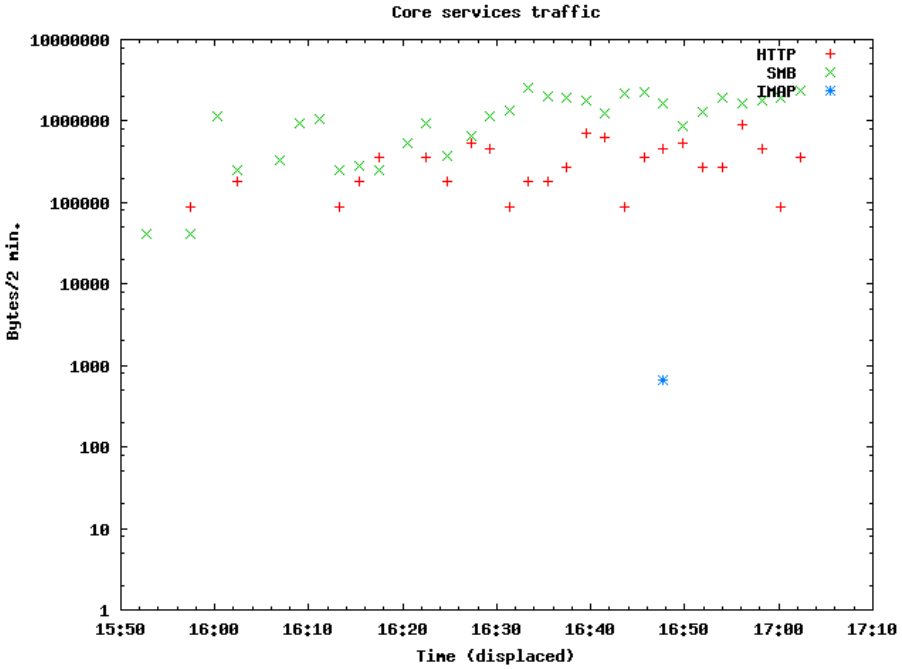

Figure 5.8: The distribution of core services from the VSIM lab, model two, with the timescale displaced by 8 hours and 10 minutes – the 15:50 marker thus reads 07:40. There seems to be a buildup of service actions, as should be when considering that this stretch of time sees the biggest shift in service use as depicted in figure 5.1.

The 24 hour simulation needed a few more testing cycles to reveal more of the potential which was hitherto hampered by stray processes.

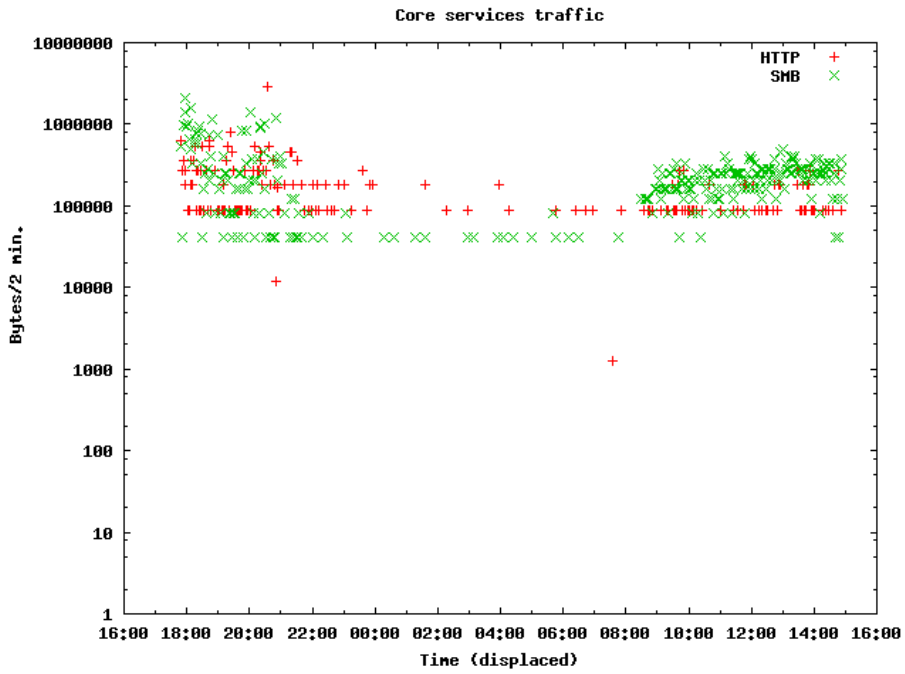

Figure 5.9: The distribution of core services from the VSIM lab, model two, for 21 hours. 10 virtual users were deployed, and intervalsec set to 10 seconds. Note that the end doesn't look as if it is going to look like the start. The reason can be that the client is slowly being encumbered by stray processes as previously explained. Also, neither IMAP nor SMTP was observed. The seemingly straight lines are probably due to the fixed size of the service actions. The service actions could therefore be improved to facilitate different sizes.

Chapter 6

Conclusions and Discussion

6.1 In conclusion

When work on the thesis begun in the beginning of January, what sparked this idea for a thesis was to find a way to implement the equivalent of the inverse of network traffic monitoring software; something which could regenerate human generated traffic with the same characteristics, if any. This lead to the questions of section 2.1 which sought answering:

- What does a normal day of user generated network traffic look like?

  A normal day of user generated network traffic was plotted in figure 5.1. Distinct characteristics were found, like how the traffic volume varied throughout the day, which closely resembled a slightly malformed sinus curve, and disproportional traffic volumes relative to different services.

- In what way can it be found out if a system solution, that is operating in a production environment, scales[1] when additional users enter the system?

  An investigation of what others have done one the subject was carried out. For the most part, benchmarking, see section 3.3, is *de facto standard*. The VSIM approach is a different one. The results show that virtual users are able to replicate the traffic patterns seen in an organization, and deploying more virtual users increases the similiarities to the real traffic. Given all the presumptions for the simulation, it looks to be a viable option to use in the process of estimating the scalability. However, more work is needed with the service action scripts and testing.

- In what way can a tool be devised to make it possible to predict the scalability of a system?

  The tools in this thesis didn't give a clear way of telling how much traffic one user would generate, but observations showed that the method was on to something by showing increased number of virtual users looking more like the real ones. Thus, the methods that was devised in section 4 provides one way of nearing the goal, and the preliminary results gives clues as to what this method can accomplish.

- In what way can user generated traffic be regenerated to such a degree that on average, there is no clear distinction?

  This was admittedly to shoot for the moon. The VSIM approach looked successful in making what could resemble user traffic, as seen in section 5.3.2,

[1] i.e. is able to cope with the added load

but there were clear distictions. Compared to real user traffic, more conform traffic consisting of the same operations all the time, were generated. It also looked as if it missed some of the noise and sudden bursts from ordinary users, but again this could be because too few virtual users were deployed. It should be possible to fix this in future versions. It could also be tested to see if the method is idempotent by running repeated simulations feeding the output to the input of the next to see if they converge on a fixed set of values. All in all, however, the experiment and the results show that this looks to be a step in the right direction.

- Can this be a tool for estimating when modifications of existing configurations are needed?

  The VSIM approach proved to be much work in setting up the simulation, adding users etc. It *can* be used as a tool for estimating when new hardware, or modification of existing configurations, is needed, and offers a different approach than the methods explained in section 3.3. The same goes for the way the VSIM tool were made to find out where computer systems need to be readjusted or upgraded to cope with added system load, scalability and predictability. It is possible, by using readily available and free[2] tools, to simulate real users – but at this stage it seems to be difficult to make the distinction between real and simulated traffic go away completely. The user generated traffic was more noisy, and the simulated traffic was more conform.

Using a probability based approach like this increases the chance of regenerating only the biggest services if run with a small amount of virtual users, neglecting to perform small services (like IMAP and SMTP for the OUC student net) and create bursty traffic. But with many more users, the possibility of behaving more like real users looks to increase. On account of that, more test iterations are needed.

6.2 Future work

First and foremost, more testing of both model one and particulary model two is needed. The reason for the depletion of the client host's resources needs to be found and dealt with. Secondly, the size of the data that is being sent to simulate traffic, and the number of users needs to be tuned. In so doing, the method outlined in section 5.2.5, page 53 can be done. Thirdly, the service actions could be changed so not just the same amount of traffic is generated per service action. When that has been taken care of, there are several posibilities for expanding the VSIM experiment:

- Introduce splay time to make the virtual users not execute at exactly the same time would be beneficial.
- Instead of, or in addition to using a time series as basis for model two, one could use a graph, given by e.g. a mathematical expression for a sinus curve, to get the probabilities for launching service events. The chance of running a service for an interval would be equal to calculating the area beneath the curve, from interval start to interval end along the x–axis and the curve's heights as y–axis. Integration could be used to calculate this area. This would eliminate the need for a variable controlling how often a virtual user should check for doing a service action.
- It would also be of interest to have some way of time-shrinking the simulation. This can be achieved by using the sinus–wave approach, decreasing the wavelength by changing the formula.

[2] as in the GNU or BSD license

- It could be interesting to compare matrixes generated from model one from different organizations, and also to further develop it to make distinctions between fine–grained operations such as read and write to a network file service.

Bibliography

[1] W. Richard Stevens. Tcp/ip illustrated: The protocols volume 1. *Addison Wesley*, 1994.

[2] M. Burgess. Theoretical system administration. *Proceedings of the Fourteenth Systems Administration Conference (LISA XIV) (USENIX Association: Berkeley, CA)*, page 1, 2000.

[3] H. Haugerud and S. Straumsnes. Simulation of user-driven computer behaviour. *Proceedings of the Fifteenth Systems Administration Conference (LISA XV) (USENIX Association: Berkeley, CA)*, page 101, 2001.

[4] Trond Reitan Mark Burgess, Hĺk Haugerud and Sigmund Straumsnes. Measuring system normality. *ACM Transactions on Computer Systems*, 2002.

[5] George Varghese Cristian Estan, Stefan Savage. Automatically inferring patterns of resource consumption in network traffic. *ACM/SIGCOMM*, 2003.

[6] Paul Barford and Mark Crovella. Measuring web performance in the wide area. *Performance Evaluation Review*, 1999.

[7] Paul Barford and David Plonka. Characteristics of network traffic flow anomalies. *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 2001.

[8] Aiko Pras Remco Poortinga, Remco van de Meent. Analyzing campus traffic using the meter-mib. *Proceedings of Passive and Active Measurement Workshop*, 2002.

[9] John H. Howard et al. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 1988.

[10] Gaurav Banga and Peter Druschel. Measuring the capacity of a web server. *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[11] Benjamin W. Wah Pankaj Mehra. Synthetic workload generation for load-balancing experiments. *IEEE Parallel & Distributed Technology: Systems & Technology*, Volume 3 , Issue 3:4 – 19, 1995.

[12] Sameer Jayendra Shah. Network benchmarking. *California Polytechnic State University Senior Project*, 1997.

[13] Smith Hernandez-Campos and Jeffay. How real can synthetic network traffic be? *University of North Carolina at Chapel Hill*, 2004.

[14] Yuan Gao Kevin Jeffay F. Donelson Smith Michele Weigle Jin Cao, William S. Cleveland. Stochastic models for generating synthetic http source traffic. *http://www.ieee-infocom.org/2004/Papers/33 _ 2.PDF*, 2004.

[15] Evangelos P. Markatos Spyros Antonatos, Kostas G. Anagnostakis. Generating realistic workloads for network intrusion detection systems. *Proceedings of the Fourth International Workshop on Software and Performance (WOSP2004)*, 2004.

[16] Ted Gent Brendan Murphy. Measuring system and software reliability using an automated data collection process. *Quality and Reliability Engineering International*, 11, no.5:341–53, 1995.

[17] Pichaya Tandayya. A project proposal for development of a distributed interactive simulation system. *Faculty of Engineering, Prince of Songkla University*, Unpublished.

[18] Richard Bejtlich. The tao of network intrusion detection. *Addison-Wesley Professional Book series*, 2004.

[19] M. Burgess. Automated system administration with feedback regulation. *Software practice and experience*, 28:1519, 1998.

Appendix A

Some limitations encountered

Sometimes even ls had to throw in the handkerchief..:

```
pc168-132:~/lf_/analyser# ls /root/dumpdisk/lf/1304/plcdir/plcdump_00001* | wc -l
bash: /bin/ls: Argument list too long
0
```

There were a substantial number of files..

```
pc168-132:~/dumpdisk/lf/1304/plcdir# ls -al | wc -l
10729
```

..which if only a subset of needed to be counted, had to be done like this:

```
pc168-132:~/dumpdisk/lf/1304/plcdir# ls -al | grep 00001 | wc -l
9703
```

Appendix B

Some help for future work

`"tcp[2:2] >= 1000 and tcp[2:2] <= 2000" to filter on port ranges`

The first two bytes in a TCP header are the destination port number, and the following two are the source port. Therefore, we can view all destination ports between 8192 and 8294 with the following command:

```
tcpdump "tcp[0:2] >= 8192 and tcp[0:2] <= 8294"
```

B.1 Some regeneration methods tried out

These methods for generating emails were also tested:

B.1.1 Telnet

A dump of a SMTP session was resent through a telnet connection to port 25 of an email server. This resulted in an email with no subject, and there were problems since the server didn't get time to reply before the next command was sent. It's a conversation, not a one–way speech. Another peculiarity was that Mozilla Thunderbird didn't start with subject: as normal emails do, but with Message–ID:. When it was sent through Thunderbird it got it right, but the subject would have to be sent first using telnet.

B.1.2 Flowreplay

Another approach was flowreplay from the tcpreplay tool. Similarly, a dumped SMTP conversation was used:

```
root@banana:~ # flowreplay email.plc
Unable to determine TCP state for node 0xbc0a1900025cd4d8
We won't connect (128.39.89.10:25 -> 128.39.73.13:2748) on non-Syn packets
```

The email was successfully sent (again). The options were tried out to see any effect, and

```
root@banana:~ # flowreplay -m wait -w 1.0 email.plc
Unable to determine TCP state for node 0xbc0a1900025cd4d8
We won't connect (128.39.89.10:25 -> 128.39.73.13:2748) on non-Syn packets

root@banana:~ # flowreplay  -w 1.0 email.plc
```

```
Unable to determine TCP state for node 0xbc0a1900025cd4d8
We won't connect (128.39.89.10:25 -> 128.39.73.13:2748) on non-Syn packets
```

produced a authentic emails, just like the ones sent earlier.

TCPdumping mozilla firefox behavior, it was noticed that it wasn't terminating it's sessions for each http GET request, but held them up even after the browsing was finished. This may be to allow for faster surfing, since the overhead in starting a new tcp session is omitted.

Using flowreplay to replay http requests while dumping them with tcpdump, and replay the new dumped file again, showed that flowreplay continuously shaved off some bytes and packets from the dump file. This method of using flowreplay showed it didn't converge to a stable state of sending the same amount of packets. After three iterations, the packet count went from 182 to 85 to 74 to 69. This was shown when trying to replay the dump with 74 packets:

```
root@banana:~ # flowreplay -m 'bytes' http_hio_replayed_2.plc
We won't connect (158.36.78.2:80 -> 128.39.73.41:45819) on non-Syn packets
We won't connect (128.39.73.41:45819 -> 158.36.78.2:80) on non-Syn packets
We won't connect (158.36.78.2:80 -> 128.39.73.41:45819) on non-Syn packets
We won't connect (128.39.73.41:45819 -> 158.36.78.2:80) on non-Syn packets
We won't connect (158.36.78.2:80 -> 128.39.73.41:45819) on non-Syn packets
We won't connect (128.39.73.41:45819 -> 158.36.78.2:80) on non-Syn packets
```

and replaying that again:

```
root@banana:~ # flowreplay -m 'bytes' http_hio_replayed_3.plc
We won't connect (158.36.78.2:80 -> 128.39.73.41:45826) on non-Syn packets
We won't connect (128.39.73.41:45826 -> 158.36.78.2:80) on non-Syn packets
```

(55 packets) - then (surprisingly) 67,67,53,64,64,65,45,46,59 packets after 12th iteration. Investigation using Ethereal revealed inconclusive evidence, but it seemed that fewer and fewer of the server replies were found, in the end only the ones from the start of the session. But the requests were left.

Flowreplay also Resat the connection, unlike mozilla.

Doing the same with TCPreplay showed that more packets were generated with more iterations. 182–195–206.

Conclusion is to use the first captured tcpdumped data to be replayed with flowreplay. Flowreplay is in alpha version so it's understandable.

B.2 Optimizing testbed

The kernel of the dumphost was tuned to better be able to receive packets, following the guidelines at http://www.samag.com/documents/s=8920/sam0311a/0311a.htm and the sysctl manpage. The rule of thumb is to have large network buffers for large transfers, and smaller and more numerous for small but many transfers. As there would probably be a lot of packets, the buffers were increased in size:

```
net.core.rmem_default = 223232
net.core.rmem_max = 446464
net.ipv4.tcp_rmem = 4096 131072 262144
```

These are variables used to control the size of the receive buffer. They can make a large impact especially on a client system, as well as for file servers.

In addition, the hard drive was tuned with hdparm:

```
hdparm -c1 /dev/hda
/etc/hdparm.conf
```

One could consider putting the driver for the capture NIC in NAPI mode. That uses polling rather than interrupts, making it less efficient under light loads, but better under sustained heavy loads.

Appendix C

Availability

The VSIM project with scripts and sample data can be downloaded from http://www.iu.hio.no/~s113111/vsim/.